Contents

SOFTWARE ENGINEERING (IV-SEM., CS BRANCH)

PAGE NO.

UNIT-I : The Software Product and Software Process

Software Product and Process Characteristics .... (03 to 12)

Software Process Models – Linear Sequential Model, Prototyping Model, RAD Model, Evolutionary Process Models like Incremental Model, Spiral Model, Component Assembly Model, RUP and Agile processes, Software Process customization and improvement, CMM .... (13 to 36)

Product and Process Metrics .... (36 to 54)

UNIT-II : Requirement Elicitation, Analysis and Specification

Functional and Non-functional requirements, Requirement Sources and Elicitation Techniques .... (55 to 64)

Analysis Modeling for Function-oriented and object-oriented software development, Use case Modeling .... (64 to 83)

System and Software Requirement Specifications, Requirement Validation, Traceability .... (83 to 98)

UNIT-III : Software Design

The software Design Process, Design Concepts and Principles .... (99 to 110)

Software Modeling and UML, Architectural Design, Architectural Views and Styles, User Interface Design, Function-oriented Design, SA/SD Component Based Design, Design Metrics .... (110 to 152)

UNIT-IV : Software Analysis and Testing

Software Static and Dynamic analysis, Code inspections .... (153 to 157)

Software Testing Fundamentals, Software Test Process, Testing Levels, Test Criteria, Test Case Design, Test Oracles, Test Techniques, Black-Box Testing, White-Box Unit Testing and Unit Testing Frameworks, Integration Testing, System Testing and other Specialized Testing, Test Plan, Test Metrics, Testing Tools .... (157 to 198)

Introduction to Object-oriented analysis, design and comparison with structured software engineering .... (198 to 202)

UNIT-V : Software Maintenance and Software Project Measurement

Need and Types of Maintenance, Software Configuration Management (SCM), Software Change Management, Version Control, Change control and Reporting .... (203 to 213)

Program Comprehension Techniques, Re-engineering, Reverse Engineering, Tool Support .... (213 to 223)

Project Management Concepts, Feasibility Analysis, Project and Process Planning, Resource Allocations .... (224 to 233)

Software efforts, Schedule and Cost estimations, Project Scheduling and Tracking, Risk Assessment and Mitigation, Software Quality Assurance (SQA), Project Plan, Project Metrics .... (234 to 264)

UNIT 1

THE SOFTWARE PRODUCT AND SOFTWARE PROCESS

SOFTWARE PRODUCT AND PROCESS CHARACTERISTICS

Q.1. What is a software product ? Also discuss its types.

Ans. Software products are nothing but software systems delivered to customer with the documentation that describes how to install and use the system. In certain cases, software products may be part of system products where hardware as well as software is delivered to a customer.

Software products fall into two broad categories –

(i) Generic Products – These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them.

(ii) Customized Products – These are systems that are commissioned by a particular customer. Some contractor develops the software for that customer.

Q.2. Distinguish between generic and customized software products. Which one would generate more revenue for a company ? Give the reasons behind your answer. ***(R.G.P.V., June 2012)***

Ans. An important difference between generic and customized software products is that, in generic products, the organization which develops the software controls the software specification. In contrast, for custom products, the specification is usually developed and controlled by the organization that is buying the software.

The generic software will generate more revenue for a company because it is created to sell on the open market to any customer who can buy them.

Q.3. Explain the characteristics of the software. ***(R.G.P.V., June 2016)***

Ans. Following are the characteristics of software –

(i) Software is developed (or engineered) rather than manufactured, in the classical sense.

Although having some similarities, software development and hardware manufacture are two fundamentally different activities. In both, high quality can only be achieved through good design. But there may introduce quality problems in the design phase of hardware, which are nonexistent for software. Both of them are human dependent, but in both, the relationship between the people involved and work accomplished is entirely different. Although both require a "product" but with different approaches.

Software costs are related with the applied engineering, which directly means that software projects cannot be managed as if they were manufacturing projects.

(ii) Software doesn't wear out, but it does deteriorate.

Observe the fig. 1.1, which depicts the failure rate as a time function, for hardware. The relationship shown in fig. 1.1 is often called the ***bathtub curve.*** It indicates the relatively high failure rates of hardware in its early life, which are often attributable to design or manufacturing defects. For some time, defects are corrected and the failure rate drops to a ***steady-state*** level. But as the time passes, the failure rate again rises due to the suffering of hardware components from the cumulative affects of dust, vibration, abuse, temperature extremes and many other atmospheric maladies. Simply, we can say the hardware begins to wear out.

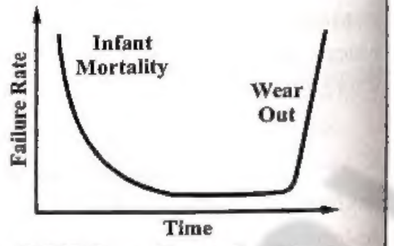


Fig. 1.1 Failure Curve for Hardware

Unlike hardware, software is not susceptible to the atmospheric maladies. Theoretically, therefore, the failure curve for software should be somewhat idealized (see fig. 1.2). Early in the life of a program, undiscovered defects will cause high failure rates. However, these defects are corrected, without introducing other errors and the curve flattens. This idealized curve is a gross over-simplification of actual failure models for software. Thus, the implication is clear, i.e., software doesn't wear out, but it does deteriorate.

Consider the actual curve shown in fig. 1.2. We find that, during its life, software will undergo changes. And due to these changes, it is highly probable that some new defects will be introduced, causing the curve to spike (see fig. 1.2). Before returning to the original steady-state failure rate, the curve spikes again due to the another requested change. As a result, the minimum failure rate level begins to rise slowly. Thus, we can say, the software is deteriorating due to change.

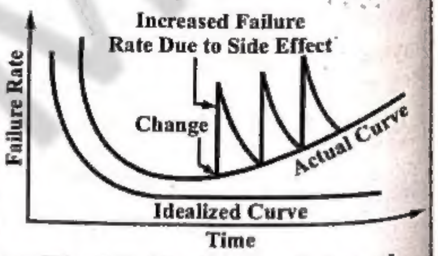


Fig. 1.2 Idealized and Actual Failure Curves for Software

(iii) Most software continues to be custom built. Although, however, the industry is moving toward component-based assembly.

Elaborating this point, consider the manner in which the control hardware for a computer-based product is built. The engineer simply draws a schematic of needed digital circuitry, does some basic analysis for the sake of proper functioning assurance and then search for the catalogs of digital components.

With the evolution of an engineering discipline, a collection of standard design components is created. Only two of the thousand standard components, i.e., standard screws and off-the-shelf integrated circuits, are used by mechanical and electrical engineers during the design of new systems.

A software component should be designed and implemented such that it can be rushed in several different programs. In the early years of development, scientific subroutine libraries were built. They were reusable in a broad array of engineering and scientific applications.

Q.4. Define software engineering. Explain software engineering – A layered technology. ***(R.G.P.V., May 2018)***

Ans. Software engineering is the engineering whose goal is the cost-effective development of system software. It concerned the production of software from the early stages of system specification to maintaining of the system after it has gone into use. There are two key phrases of the software engineering.

(i) Engineering Discipline – Engineers apply theories, methods and tools to make the things to work. They apply their theories on appropriate and selected things and always try to discover solutions to problems, if there are no applicable theories and methods to support them.

(ii) All Aspects of Software Production – Software engineering is not only concerned with technical processes of software development. It also concerned with the activities such as software project management and also with the development of tools, methods and theories which are helpful to support software production.

A layered technology of software engineering is divided into four layers –

(i) Quality focus
(ii) Process
(iii) Methods
(iv) Tools.

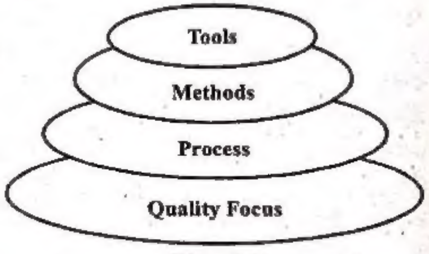


Fig. 1.3

(i) Quality Focus –

(a) Any engineering approach must rest on an quality.

(b) The 'Bed Rock' that supports software engineering is quality focus.

(ii) Process –

(a) Foundation for software engineering is the process layer.

(b) Software engineering is the process that holds all the technology layers together and enables the timely development of computer software.

(c) It forms the base for management control of software project.

(iii) Methods –

(a) The technical questions are provided by the software engineering methods for building software.

(b) Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

(iv) Tools –

(a) The software engineering tools provide automated or semi-automated support for the process and the methods.

(b) Tools are integrated so that information created by one tool can be used by another.

Q.5. Explain how do the use of software engineering principles help to develop software products cost effectively and timely. Elaborate your answer by using suitable example. ***(R.G.P.V., June 2013)***

Ans. Software engineering principles deploy mainly two important principles to resolve the problems arising due to human cognitive limitations. These two principles are abstraction and decomposition. A simple understanding of the human cognition mechanism would give us an insight into why the exploratory style of development leads to undue increase in the time and effort needed to develop a programming solution. It can also explain why it quickly becomes practically infeasible to sort out problems larger than a certain size. Whereas, using software engineering principles, the needed effort expands almost linearly with size.

Abstraction refers to construction of a simpler version of a problem by ignoring details. The principle of abstraction is popularly known as modelling. To use the principle of abstraction to understand a complex problem, we must concentrate on only one aspect of the problem each time, and avoid details that are not in accordance with the aspect we are focusing. Whenever we omit some details of a problem, we get a model of the problem. In our day-to-day life, we use the principle of abstraction frequently to understand a problem well or to evaluate a situation. Consider the following example – suppose you are asked to develop an understanding of all the living beings inhabiting the earth. If you use the naive approach, you would begin taking up one living being after another who inhabit the earth and begin understanding them. Even after putting in tremendous effort, you would make little progress and left confused since there are billions of living things on earth and the information just too much for any one to handle. Instead, what can be done is to build and understand an abstraction hierarchy of all living beings. At the top level, we understand that there are three basically different types of living beings, i.e., plants, animals, and fungi. Slowly more details are added about each type at each successive level, until we reach the level of the different species at the leaf level of the abstraction tree.

The principle of decomposition advocates decomposing the problem into various small individual parts. The small parts are then taken up one by one and solved independently. The rationale behind this is that each small part would be easy to understand and can be easily solved. The whole problem is solved when all the parts are solved. Consider the following problem which shows the use of the decomposition principle. You would understand a book better when the contents are decomposed into more or less individual chapters. That is, each chapter focuses on a separate topic, rather than when the book combines up all topics together throughout all the pages. Similarly, each chapter should be decomposed into sections such that each section focuses on different issues. Each section should be decomposed into subsections, and so on.

Q.6. Define software process. ***(R.G.P.V., June 2016)***

Ans. A software process is a set of activities, together with ordering constraints among them, such that if the activities are performed properly and in accordance with the ordering constraints, the desired result is produced. The desired result is the high-quality software at low cost.

Q.7. What do you mean by a software process ? What is the difference between a methodology and a process ? Explain using suitable examples. ***(R.G.P.V., June 2013)***

Or

What is software process ? How the process is different from methodology show by an example ? ***(R.G.P.V., June 2015)***

Ans. **Software Process** – Refer to Q.6.

Difference between Methodology and a Process – A process includes all the activities beginning from product inception through delivery and retirement. In addition, it resolves issues such as ordering of the activities, reuse, documentation, testing, parallel work performed by team members and coordination with the customer, etc. Whereas, a methodology includes only a single or at best a few individual activities involved in the development.

For example, testing methodology, design methodology, etc. point out a single activity in a life cycle. Thus, we may think that a process scales up a methodology to the entire life cycle.

Q.8. Describe the important characteristics of a software process.
(R.G.P.V., Dec. 2004)

Ans. The main objective of process is same as that of software engineering namely optimality and scalability. Optimality defines the process should be able to produce high-quality software at low cost, while scalability means that it should also be applicable for large software projects. So, process should have some desirable characteristics to achieve the objective, which are as follows –

(i) Predictability – Predictability can be considered as a fundamental property of any process. Predictability of a process determines how accurately the outcome of following process in a project can be predicted before the project is completed. If a process is not predictable, then it is of limited use.

Similarly, quality prediction is that quality of the product is determined largely by the process used to develop it. On this basis, quality of the product of a project can be estimated or predicted by seeing the quality of the product that has been produced in the past by the process being used in the current project. So effective management of quality assurance activity largely depends on the predictability of the process.

A predictable process is under statistical also called as under statistical control. Statistical control implies the prediction for a process, which are generally based on the past performance of the process, are only probabilistic.

So, if one who wants to consistently develop software of high quality at low cost, it is necessary to have a process that is under statistical control. A predictable process is an essential requirement for ensuring good quality and low cost.

(ii) Testability and Maintainability – One of the important objectives of the development project should be to produce software that is easy to maintain and the process should be such that it ensures this maintainability.

The second important effort is the testing, which consumes most resources during development. Underestimating the testing effort often causes the planner to allocate insufficient resources for testing, which in term, results in unreliable software or schedule slippage.

The goal of the process should not be to reduce the effort of design and coding, but to reduce the cost of testing and maintenance. Both testing and maintenance depend heavily on the design and coding of software, and these costs can be considerably reduced if the software is designed and coding to make testing and maintenance easier.

(iii) Early Defect Removal and Defect Prevention – In this section the main moral is that, we should attempt to detect errors that occur in a phase during that phase itself and should not wait until testing to detect errors. In reality, sometimes testing is the sole point where errors are detected. Besides the cost factor, reliance on testing as the primary source for error detection due to the limitations of testing, will also result in unreliable software. Error detection and correction should be a continuous process throughout the software.

The detecting errors have been introduced is an objective by the process. However, even better is to provide support for detect prevention. The defect removal methods that exist today are limited in their capability and they cannot detect all the defects that are introduced. So, the cost of defect removal is generally high, particularly if they are not detected for a long time, then to reduce the total number of residual defects that exists in a system at the time of delivery and to reduce the cost of defect removal, an obvious approach is to prevent defects from getting introduced, it requires the process of performing the activities should be such that fewer defects are introduced. The method support defect prevention is to use the development process to learn and so the methods of performing activities can be improved.

(iv) Process Improvement – The process is not a static entity. The fundamental goals of any engineering discipline are improving the quality and reducing the cost of products. In software, as the productivity and quality are determined by the process, to satisfy the engineering objectives of quality improvement and cost reduction, the software process must be improved. Having process improvement as a basic goal of the software process implies that the software process used is such that it supports its improvement. This needs that there be means for evaluating the existing process and understanding the weaknesses in process.

Q.9. Explain software process. Enumerate the activities common to all software processes. Also list the characteristics of software processes.
(R.G.P.V., June 2011)

Or

Explain in detail about the software process. (R.G.P.V., May 2019)

Ans. **Software Process** – Refer to Q.6.

There are four fundamental process activities which are common to all software processes. These activities are –

(i) Software Specification – The functionality of the software and constraints on its operation must be defined.

(ii) Software Development – The software to meet the specification must be produced.

(iii) Software Validation – The software must be validated to ensure that it does what the customer wants.

(iv) Software Evolution – The software must evolve to meet changing customer needs.

Characteristics of Software Process – Refer to Q.8.

Q.10. Explain unified process. What are its characteristics ?
(R.G.P.V., Dec. 2014)

Ans. The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).

The ***unified software development process*** represents a number of component-based development models being proposed in the industry. The unified process, using the ***unified modeling language (UML)***, defines the system building components and the component connecting interfaces. And by using a combination of iterative and incremental development, it defines the system function by using scenario-based approach. It then couples function with an architectural frame work that identifies the form the software will take.

The unified modeling language was developed in conjunction with the unified process. Throughout the entire unified process lies the idea of creating models of the system being constructed. Models represent abstract views of the system from a particular point of view. These models are captured and communicated using UML.

The unified pocess is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The rational unified process is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP and so the names tend to be used interchangeably.

The name Unified Process as opposed to Rational Unified Process is generally used to describe the generic process, including those elements which are common to most refinements. The Unified Process name is also used to avoid potential issues of copyright infringement since Rational Unified Process and RUP are trademarks of IBM. The first book to describe the process was titled The Unified Software Development Process (ISBN 0-201-57169-2) and published in 1999 by Ivar Jacobson, Grady Booch and James Rumbaugh. Since then various authors unaffiliated with Rational Software have published books and articles using the name Unified Process, whereas authors affiliated with Rational Software have favored the name Rational Unified Process.

The Unified Process identifies core workflows that occur during the software development process. These workflows include Business Modeling, Requirements, Analysis, Design, Implementation and Test. The workflows are not sequential and likely will be worked on during all of the four phases. The workflows are described separately in the process for clarity but they do in fact run concurrently.

The Unified Process has three distinguishing characteristics. These characteristics are –

(i) Use-Case Driven – The process employs Use Cases to drive the development process from inception to deployment.

(ii) Architecture-Centric – The process seeks to understand the most significant static and dynamic aspects in terms of software architecture. The architecture is a function of the needs of the users and is captured in the core Use Cases.

(iii) Iterative and Incremental – The process recognizes that it is practical to divide large projects into smaller projects or mini-projects. Each mini-project comprises an iteration that results in an increment. An iteration may encompass all of the workflows in the process. The iterations are planned using Use Cases.

Q.11. Differentiate between a software product and a software process.
(R.G.P.V., Dec. 2004, June 2012)

Ans. Following are the differences between a software product and a software process –

S.No.	*Software Product*	*Software Process*
(i)	Software product is the computer software that software engineers design.	Software process is the sequence of steps that are followed to develop a timely and high quality software product.
(ii)	It includes programs which can run on any computer, documents in the form of hard copy and virtual forms and relevent data and pictures.	It defines a framework for a set of key process areas (KPAs) that must be determined for effective delivery of software engineering technology.
(iii)	It is the main driving force that drives business decision making. It works as the basis for modern scientific investigation and engineering problem solving.	It offers stability, control and organization to an activity that can if left uncontrolled, become quite chaotic.
(iv)	It is built by applying software engineering approach, by applying a process that leads to a high quality software.	Choosing a process relies upon the software you are building as one process might be suitable for creating software for aircraft avionics system while an entirely different process would be needed for the creation of website.
(v)	Software product relies on software process for its stability quality and control.	It is more important than software product.

Q.12. Discuss the relationship between processes, projects and products.

Ans. A software process specifies a method of developing software. A software project, on the other hand, is a development project in which a software process is used. And software products are the outcomes of a software project. Each software development project starts with some needs and ends with some software that satisfies those needs. A software process specifies the abstract set of activities that should be performed to go from user needs to final product. The actual set of executing the activities for some specific user needs is a software project. And all the outputs that are produced while the activities are being executed are the products. The software process can be viewed as an abstract type, and each project is done using that process as an instance of this type. In other words, there can be many projects for a process and there can be many products produced in a project. This relationship is shown in fig. 1.4.

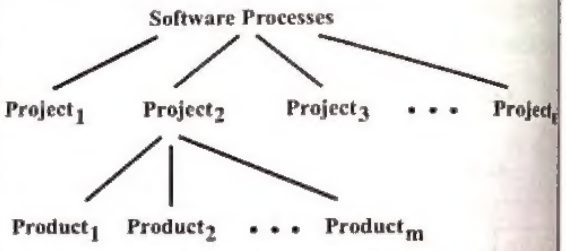


Fig. 1.4 Processes, Projects and Products

Now a question arises, if the sequence of activities is provided by the process, what is the difficulty in following it in a project ? First, the sequence of activities specified by the process is typically at an abstract level because they have to be usable for a wide range of projects. Hence, "implementing" them in a project is not straight-forward. In a sense, the process provides a "checklist", with an ordering constraint.

Overall, the process specifies activities at an abstract level that are not project-specific. It is a generic set of activities that does not provide a detailed roadmap for a particular project. The detailed roadmap for a particular project is the ***project plan*** that specifies what specific activities to perform for this particular project, when and how to ensure that the project progresses smoothly.

It should be clear that it is the process that drives a project. A process limits the degrees of freedom for a project by specifying what types of activities must be done and in what order. Further, restrictions on the degrees of freedom for a particular project are specified by the project plan, which, in itself, is within the boundaries established by the process. With this, the hope is that one has the "shortest" path from the user needs to the software satisfying these needs.

As each project is an instance of the process it follows, it is essentially the process that determines the expected outcomes of a project. Due to this the focus of software engineering is heavily on the process.

SOFTWARE PROCESS MODELS – LINEAR SEQUENTIAL MODEL, PROTOTYPING MODEL, RAD MODEL, EVOLUTIONARY PROCESS MODELS LIKE INCREMENTAL MODEL, SPIRAL MODEL, COMPONENT ASSEMBLY MODEL, RUP AND AGILE PROCESSES, SOFTWARE PROCESS CUSTOMIZATION AND IMPROVEMENT, CMM

Q.13. What is a model ? What are the benefits with model ? ***(R.G.P.V., Dec. 2015)***

Ans. A model is a representational abstraction containing a set of logical and quantitative relationships between the members of a set of variables or primitives. An important reason behind constructing a model is that it helps to manage the complexity in a problem and facilitates arriving at good solutions and at the same time helps to reduce the design costs. A model helps to understand the problem easily.

Q.14. What is software development life cycle model ? Why is it important to adhere to a life cycle model while developing a large software product ? ***(R.G.P.V., June 2010)***

Or

Explain in detail about the life cycle process. ***(R.G.P.V., Dec. 2017)***

Ans. A life cycle model specifies the different activities that need to be performed to develop a software product and the sequencing of these activities. The software life cycle is also sometimes called as the systems development life cycle (SDLC). Basically, the classical waterfall model is the basic life cycle model. Every software product starts with a request for the product by the customer. This is called ***product conception***. Starting with this stage it undergoes transformations through a series of identifiable stages until it is fully developed and released to the customer. After release, the product is used by the customer and is finally retired when it is no longer useful. This forms the essence of the life cycle of every software product. Life cycle is not strange to software development. In fact, each business organizations conducts its business through a certain sequence of well-defined steps. Likewise, manufacturing industries use some steps to produce their product. The software life cycle can be viewed as the business process for software development, and therefore, a software life cycle is also often called as a ***software process***.

Traditionally, the feasibility study is the first stage in the life cycle of any software product. The subsequent stages include – requirement analysis and specification, design, coding, testing and maintenance. Each of these stages is referred to as a life cycle phase. During each life cycle phase, usually several kinds of activities need to be performed and several documents produced before the end of the phase. A software life cycle model is a diagrammatic and descriptive representation of the software life cycle. A life cycle model maps the various activities conducted on a software product from its inception to retirement into a set of life cycle phases. Different life cycle models may map the fundamental development activities to phases in different ways. Thus, no matter which life cycle model is used, the basic activities are included in all life cycle models, though the activities may be performed in different order in various life cycle models.

Software development organizations have felt that adherence to an appropriate well-defined life cycle model helps to produce good quality products and that too without time and cost overruns. The main benefit of adhering to a life cycle model is that it enables development of software in a systematic and disciplined way.

Q.15. What do you understand by term life cycle model of software development ? What problems might a software development organization face if it does not follow any life cycle model during development of a large software product ? ***(R.G.P.V., June 2014)***

Or

Explain the term software development life cycle model. What are the consequences if an organization does not follow any life cycle model during development of the software product ? ***(R.G.P.V., June 2016)***

Ans. **Life Cycle Model of Software Development** – Refer to Q.14.

Problems – The main benefit of adhering to a life cycle model is that it encourages development of software in a systematic and disciplined manner. As we know that when a program is developed by a single programmer, he has the freedom to decide the exact steps through which he will develop the program. However, when a software product is developed by a team, it is necessary to have precise understanding among the team members as to – when to do what. Otherwise, if each member is allowed to do whatever activity he feels like doing, then we have the perfect recipe for chaos and project failure. Let us try to illustrate this situation using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to then in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might start doing the design for the parts assigned to him. This is one of the perfect recipes for project failure. Believe it or not, this is exactly what has caused many project failures in the part. It is not difficult to guess the reasons for the failure. Severe problems arise in interfacing the different parts and in managing the overall development. Also, consider the situation in which an engineer first develops the code, while another develops the requirements specification and design, and writes the code later. Then, the first engineer has to wait (idle) until the second engineer completes his coding before the two pieces of code can be integrated together. The use of a suitable life cycle model is crucial to the successful completion of a project.

Q.16. What is a software engineering paradigm ?

Or

Explain about software engineering paradigm in detail. ***(R.G.P.V., May 2019)***

Ans. A software engineer or a team of engineers must incorporate a development strategy that include the process, methods, and tools layers and the generic phases to solve actual problems in an industry setting. This strategy is often called as a ***process model*** or a ***software engineering paradigm***.

That is, a software process model is an abstraction of a software process. A process model relies on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

Therefore, it is essential to define process model for each software project. IEEE defines a process model as "a framework containing the processes, activities, and tasks involved in the development, operation and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use".

The various process models are –

(i) Linear sequential model or waterfall model
(ii) Prototyping model
(iii) RAD model
(iv) Evolutionary process model
(v) Incremental model
(vi) Spiral model
(vii) Component-assembly model.

Q.17. Explain waterfall model.

Or

Explain various phases of classic life cycle model.

Or

Explain linear sequential model for software development. ***(R.G.P.V., May 2018)***

Or

What are the major phases in the waterfall model of software development ? Which phase consumes the maximum effort for developing a typical software product ? ***(R.G.P.V., June 2014)***

Ans. The waterfall model or the classic life cycle is sometimes called the linear sequential model. It suggests a systematic approach to software development that starts at the system level and passes through analysis, design, coding, testing and support. The principal stages of the model as shown in fig. 1.5 are explained as follows –

(i) Requirements Analysis and Definition – The system's constraints, services and goals are determined by consultation with system users. They are then defined in detail and work as a system specification.

(ii) System and Software Design – The requirements are divided by the systems design process to either hardware or software systems. The system design process establishes an overall system architecture. Software design includes identifying and describing the basic software system abstractions and their relationships.

(iii) Implementation and Unit Testing – In this stage the software design is considered as a set of programs or program units. Unit testing involves verifying that each unit satisfies its specifications.

(iv) Integration and System Testing – The individual program units or programs are integrated and tested as a complete system to guarantee that the software requirements have been satisfied. The software system is provided to the customer after testing.

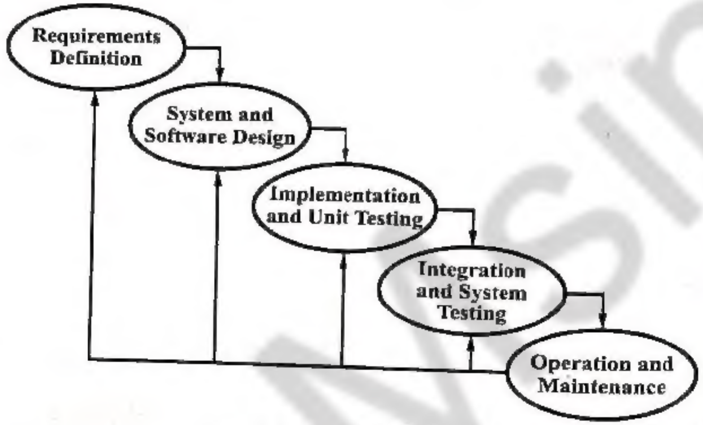

Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Fig. 1.5 Waterfall Model

(v) Operation and Maintenance – It is the longest life-cycle phase. The system is installed and used practically. Maintenance includes correcting errors which were not found in earlier stages of the life cycle, enhancing the implementation of system units and improving the system's services as new requirements are found.

Q.18. What are the advantages and disadvantages of waterfall model ?

Or

List out the Pros and Cons of waterfall model of software development.

(R.G.P.V., June 2015)

Ans. **Advantages –**

(i) Relatively simple to understand.

(ii) Each phase of development proceeds sequentially.

(iii) Allows managerial control where a schedule with deadlines is set for each stage of development.

(iv) Helps in controlling schedules, budgets and documentation.

Disadvantages –

(i) Requirements need to be specified before the development proceeds.

(ii) Changes of requirements in later phases of the waterfall model cannot be done. This implies that once an application is in the testing phase, it is difficult to incorporate changes.

(iii) No user involvement and working version of the software is available when the software is developed.

(iv) Does not involve risk management.

(v) Assumes that the requirements are stable and are frozen across the project span.

Q.19. Consider the assertion – The classical waterfall model is an idealistic model.

(i) Justify why the above assertion is true.

(ii) Even if the classical waterfall model is an idealistic model, is there any practical use of this model at all ? Explain your answer.

(R.G.P.V., June 2013)

Ans. (i) The classical waterfall model is an idealistic model because it assumes that no error is ever committed by the engineers during any of the life cycle phases, and therefore, leaves no scope for error correction. This is not a practical model in the sense that it cannot be used in real software development projects. That is, this model can be assumed as a theoretical way of developing software.

(ii) All other life cycle models are in some way depend on the classical waterfall model. Therefore, we have to first understand the classical waterfall model well, in order to be able to appreciate and develop proper understanding of other life cycle models. This model is generally adhered to for developing software documentation.

Q.20. Compare and contrast the important life cycle models.

(R.G.P.V., Dec. 2010)

Ans. The l asic software development life cycle model is classical waterfall model and all other models are considered as embellishments of this basic model.

For practically development projects, classical waterfall model cannot be us because no mechanism is provided by this model to correct the errors that a committed during any of the phases but detected at a later phase. To overco this problem, iterative waterfall model is introduced by using the provision o feedback paths. This is probably the most widely used software developme model so far. The iterative waterfall model is easy to use and understand. Th model is suitable only for well-understood problems. However, projects th suffer from many types of risks and large in size, this model is not suitable.

For the situations where projects for which the user requirements or th underlying technical aspects are not well understood prototyping model is be suited, however all the risks can be identified before the projects starts. Fo development of the user interface parts of projects, this model is very popul

For large problems, evolutionary approach is more suitable. In this mode large problems can be decomposed into a set of modules for incremen development and delivery. This model is also used widely for object-oriente development projects. However, this model can also be used in situatio where incremental delivery of the system is acceptable to the customer.

The spiral model encompasses all other life models and considered to b a metal model. Flexibility and risk handling are inherently built into this mode This model is suitable for development of technically challenging and larg software products that are prone to various kinds of risks and difficult t anticipate at the initiation of the project. As compare to other life cycle mode this model is much more complex.

Q.21. What are the characteristics to be considered for the selection o a life cycle model ?
(R.G.P.V., Dec. 201

Ans. The parameters needed to select the process model are –

(i) Project Type and Associated Risks – One of the key features o choosing a process model is to understand the project in terms of size complexity, funds available, and so on. It is very important to choose a accurate process model for the project. Besides, the risks associated with th project should also be considered.

(ii) Requirements of the Project – The most essential feature of an process model is to understand the requirements of the project. The develope software leads to ineffective systems if the requirements are not clearly define by the user or poorly understood by the developer. Hence, the requirements o the software should be very clear before choosing any process model.

(iii) A software is developed for the user. Therefore, the users shoul be consulted while choosing the process model. If users are involved in selectin the process model, the comprehensibility of the project increases. It is possibl that a user is aware of the requirements or has some idea of the requirement

Q.22. What is prototyping model ? Under what circumstances it is beneficial to construct a prototyping model ? Does the construction of a prototyping model always increase the overall cost of software development ?
(R.G.P.V., June 2003, 2014)

Or

How prototype model solve the problems over the waterfall model ?
(R.G.P.V., Dec. 2015)

Ans. The goal of a prototype model is to counter the limitations of waterfall model. The basic idea here is that instead of freezing the requirements before any design or coding can proceed, a throwaway prototype is built to help understand the requirements. As shown in fig. 1.6, the prototype model starts with requirements gathering. Developer and customer together establish the objectives of the software, identify the requirements known and outline areas where further definition is necessary. A "quick design" then takes place. The quick design emphasizes on a representation of those aspects of the software that will be visible to the customer/ user. The quick design leads to the construction of a prototype. Then, the customer evaluates the prototype and refines requirements for the software to be developed. The prototype works as a mechanism for identifying software requirements. If a working prototype is made, the developer tries to use existing program fragments or applies tools that enable working programs to be produced quickly.

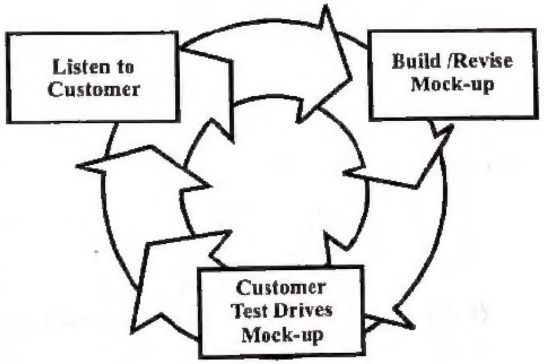


Fig. 1.6 Prototype Model

Although having some difficulties in its implementation, prototyping can be an effective model for software engineering. For having its effectiveness, one should define the rules in the beginning i.e., both the customer and the developer must agree that the prototype is built as a mechanism for defining requirements. Afterwards it is discarded and the actual required software can be engineered with concentrating on quality and maintainability.

For prototyping for the purposes of requirement analysis to be feasible its cost must be kept low. Consequently, only those features are included in the prototype that will have a valuable return from the user experience. Exception handling, recovery and conformance to some standards and formats are typically not included in prototypes. In prototyping, as the prototype is to be discarded, there is no point in implementing those parts of the requirements that are already well understood. Hence, the focus of the development is to include those features that are not properly understood.

Prototyping is often not used, because it is considered that developme costs may be high. However, in some cases, the cost of software developme without prototyping may be more than with prototyping. There are two maj reasons for this. First, the experience of developing the prototype might redu the cost of the later phases when the actual software development is do Secondly, in many projects the requirements are constantly changing, particula when development takes a long time. We saw earlier that changes in requireme at a later stage of development substantially increase the cost of the project. E elongating the requirements analysis phase, the requirements are "frozen" a later time, by which time they are likely to be more developed and consequen more stable. In addition because the client and users get experience with t system, it is more likely that the requirements specified after the prototype w be closer to the actual requirements. This again will lead to fewer changes in t requirements at a later time. Hence, the costs incurred due to changes in t requirements may be substantially reduced by prototyping. Hence, the cost the development after the prototype can be substantially less than the cost witho prototyping. Prototyping is well suited for projects requirements are hard determine and the confidence in the stated requirements is low.

The goal of a prototype model is to counter the limitations of waterf model. The basic idea here is that instead of freezing the requirements befo any design or coding can proceed, a throwaway prototype is built to he understand the requirements.

Q.23. Differentiate between waterfall model and prototype model.
(R.G.P.V., Dec. 201

Ans. Refer to Q.17 and Q.22.

Q.24. Explain the prototyping approaches in software process.
(R.G.P.V., Dec. 201

Or

Describe various software prototyping techniques. (R.G.P.V., May 201

Ans. The prototyping approaches in software process are as follows –

(i) Evolutionary Prototyping – This prototyping approach is base on the idea of developing an initial implementation, exposing to user comme and refining through many stages until an adequate system has been develop as shown in fig. 1.7.

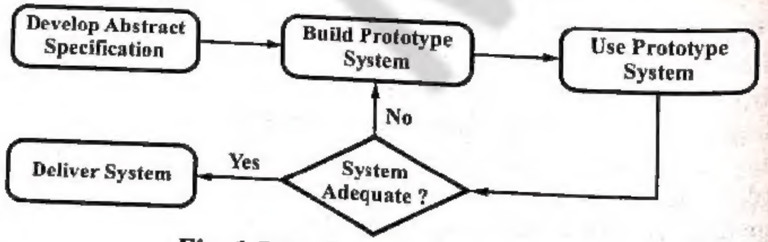


Fig. 1.7 Evolutionary Protoyping

The advantages to adopting this approach to software development are –

(a) Accelerated Delivery of the System – The software support is essential in the change of business pace. The rapid delivery and usability is more important than details of functionality or long-term software maintanability.

(b) User Engagement with the System – The involvement of user with the development process is not only to meet their requirements but it also means that the system have made a commitment to it and are likely want to make it work.

There are some problems with evolutionary prototyping which are particularly important when large, long-lifetime systems are to be developed –

(a) Management Problems – Software management is important structure for large systems to deal with a software process model that generates regular deliverables to assess progress.

(b) Maintenance Problems – This type of problems means that any one apart from the original developers are like to find it difficult to understand.

(c) Contractual Problems – The contractual model between a customer and a software developer is based around a system specification. It becomes difficult to design a contract for the system development without the system specification. The developers are unlikely to accept a fixed-price contract and they cannot control the changes requested by the end-users.

(ii) Throw-away Prototyping – This type of approach extends the requirements analysis process by reducing the overall life-cycle costs. The main function of prototype is to clarify the requirements and provide additional information for managers to assess process risks. This prototype is not used for further system development on the basis of its evaluation. The software process model based on the initial throw-away prototyping stage is shown in fig. 1.8.

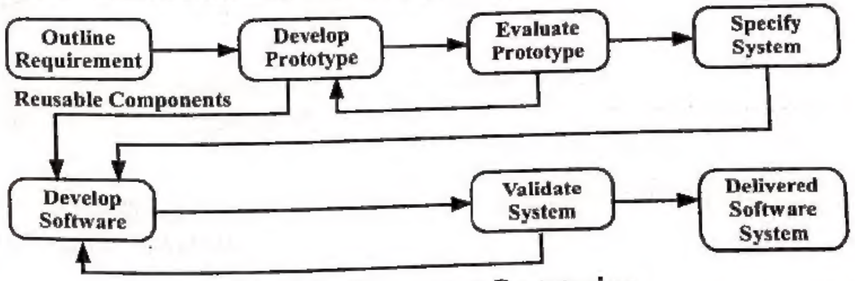


Fig. 1.8 Throw-away Prototyping

There are some problems with this approach as follows –

(a) Important features are left out of the prototype to simplify the rapid implementation. It is not possible to prototype some of the important parts of the system such as safety-critical functions.

(b) An implementation does not have any legal contract between customer and contractor.

(c) The non-functional requirements which concern reliability, robustness and safety cannot be adequately tested in prototype implementation.

Q.25. What are the advantages of developing the prototype of a system
(R.G.P.V., Dec. 201

Ans. The advantages of developing the prototype of a system are as follows.

(i) Provides a working model to the user early in the process enabling early assessment and increasing user confidence.

(ii) The developer gains experience and insight by developing a prototype, thereby resulting in better implementation of requirements.

(iii) The prototyping model serves to classify requirements, hence reducing ambiguity and improving communication between the developer and the user.

(iv) There is a great involvement of users in software development. Hence, the requirements of the users are met to the greatest extent.

(v) Helps in reducing risks associated with the project.

Q.26. What are the disadvantages of prototype model.

Ans. The disadvantages of prototype model are as follows –

(i) If the user is not satisfied with the developed prototype, then a new prototype is developed. This process goes on until a satisfactory prototype evolves. Thus, this model is time-consuming and expensive.

(ii) The developer loses focus of the real purpose of prototype and comprises on the quality of product.

(iii) Prototyping can lead to false expectations. It often creates a situation where the user believes that the development of the system is finished when it is not.

(iv) The primary goal of prototyping is rapid development. Thus, the design of the system may suffer as it is built in a series of layers without considering integration of all the other components.

Q.27. Explain prototype model. What are the advantages of developing a prototype of a system ? ***(R.G.P.V., Nov./Dec. 2007, June 2012)***

Ans. Refer to Q.22 and Q.25.

Q.28. Explain RAD model. Discuss the situation where RAD model is useful. ***(R.G.P.V., June 2010)***

Or

Explain the various phases of RAD model. Discuss the situation where RAD model is useful. ***(R.G.P.V., June 2011)***

Ans. Rapid application development is an incremental software development process model that has extremely short development cycle. It is a high speed version of the linear sequential model where rapid development is obtained by using component based construction. If requirements are well known and project scope is constrained, the RAD process enables a development team to provide a fully functional system within 60 to 90 days.

The phases of RAD approach are (see fig. 1.9) –

(i) Business modeling (ii) Data modeling
(iii) Process modeling (iv) Application generation
(v) Testing and turnover.

(i) Business Modeling – The information flow among business functions is modeled in such a manner that answers some questions as – what information is required ? What information is produced ? Who produces it ? etc.

(ii) Data Modeling – The information flow defined as part of business modeling phase is refined into data objects to help business.

(iii) Process Modeling – The data objects defined in the data modeling phase are transformed to get the information flow required to implement a business function.

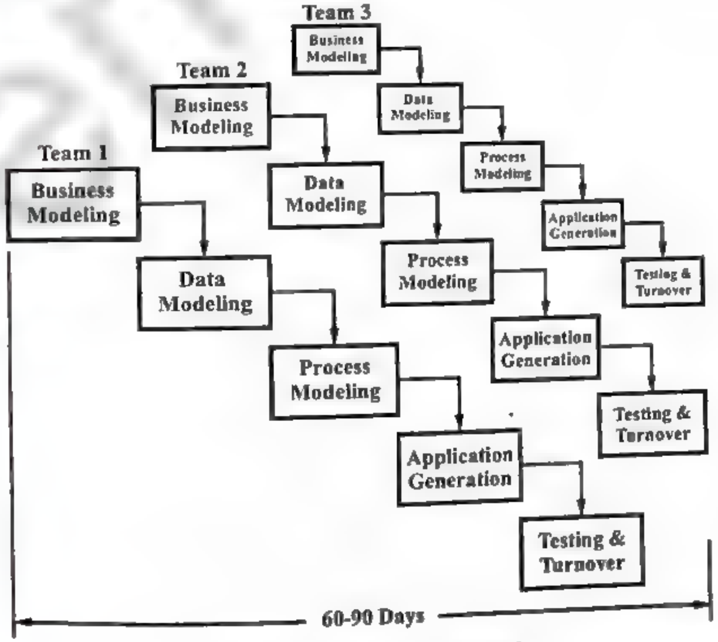


Fig. 1.9 The RAD Model

(iv) Application Generation – RAD assumes the use of fourth generation techniques to facilitate construction of software. Instead of creating software using conventional third generation programming languages the RAD process works to reuse existing program components or create reusable components.

(v) Testing and Turnover – Because the RAD process focuses reuse, many of the program components have already been tested. This decreases overall testing time.

Q.29. Define software process. Explain RAD model for software development with its various phases. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.6 and Q.28.

Q.30. What is the importance of models in software engineering? Explain with examples any two process models which are commonly used. ***(R.G.P.V., June 2017)***

Ans. Refer to Q.13, Q.14, Q.22 and Q.28.

Q.31. What are the advantages and disadvantages of the RAD model?

Ans. There are following advantages and disadvantages of the RAD model–

Advantages –

(i) Deliverables are easier to transfer as high-level abstractions, scripts, and intermediate codes are used.

(ii) Provides greater flexibility as redesign is done according to the developer.

(iii) Results in reduction of manual coding due to code generators and code reuse.

(iv) Encourages user involvement.

(v) Possibility of lesser defects due to prototyping in nature.

Disadvantages –

(i) Useful for only larger projects.

(ii) RAD projects fail if there is no commitment by the developers or the users to get the software completed on time.

(iii) Not appropriate when technical risks are high. This occurs when the new application utilizes new technology or when new software requires a high degree of interoperability with existing system.

(iv) As the interests of users and developers can diverge from single iteration to the next, requirements may not converge in RAD model.

Q.32. What are evolutionary process models ? Explain why software that are developed using evaluationary development are likely to be difficult to maintain model. ***(R.G.P.V., Dec. 2010)***

Ans. Evolutionary model depends on the idea of developing an initial implementation, exposing it to the user for refinement through many versions until an adequate system has been developed as shown in fig. 1.10. Instead of having separate specification development and validation activities, these are performed concurrently with rapid feedback across these activities.

This model of development has a number of benefits. In this model, the user gets an opportunity to experiment with a partially developed software much before the complete version of the system is produced. Therefore, the evolutionary model aids to accurately gather user requirements during the delivery of different versions of the software. Consequently, the change requests after delivery of the complete software are very less. Also, the core modules get tested thoroughly, hence decreasing chances of errors in the core modules of the complete product. Moreover, this model eliminates the requirement to commit large resources in one go for development of the system.

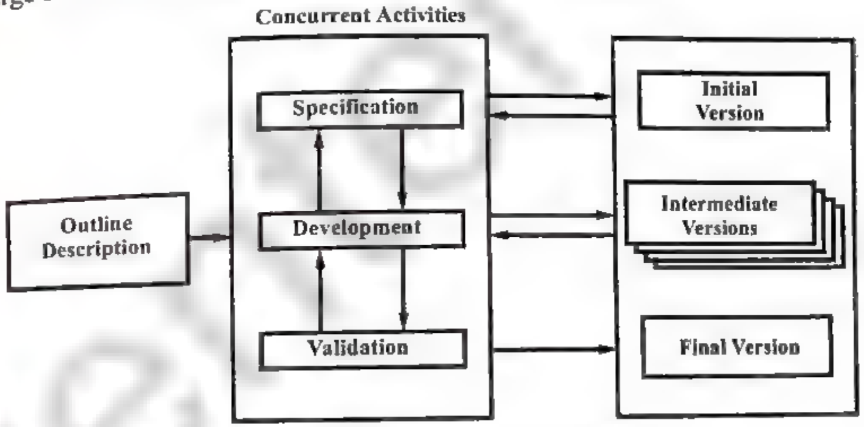


Fig. 1.10 Evolutionary Model

The main drawback of the successive versions model is that for several practical problems it is hard to split the problem into several versions that would be acceptable to the customer and which can be incrementally implemented and delivered.

There are two types of evolutionary models –

(i) Incremental model

(ii) Spiral model.

Incremental Model – The incremental model reduces rework and gives the customer an opportunity to delay decisions on their detailed requirements until they had some experience with the system.

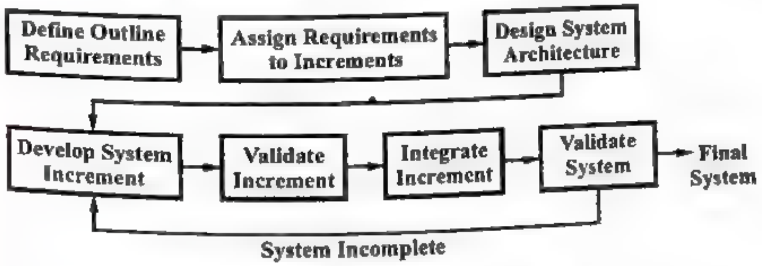


Fig. 1.11 Incremental Model

In incremental model illustrated in fig. 1.11 first the services to be provided are identified. Then a number of delivery increments are defined. Once, systems increments have been defined then requirements for the first increment are defined and that increment is developed using appropriate process. After th completion and delivery of an increment customers can put it into service. A new increments are completed they are integrated with the existing incremen so that system functionality enhances with each increment.

Continual change tends to corrupt the structure of the prototype syste This means that anyone except the original software developers is likely find it difficult to know and understand. Also, if specialized technology is use for developing rapid prototype development this may become obsolete. Thu requiring persons who have the sufficient knowledge to understand an implement the software system may be difficult.

Q.33. What are the advantages and disadvantages of incremental model

Ans. **Advantages** – Advantages of incremental model are as follows –

(i) Avoids the problems resulting in risk-driven approach in th software.

(ii) Understanding of the problem increases through successiv refinements.

(iii) Performs cost-benefit analysis before enhancing software wit capabilities.

(iv) Incrementally grows in effective solution after each multipl iteration.

(v) Does not involve a high complexity rate.

Disadvantages – Disadvantages of incremental model are as follows –

(i) Requires planning at the management and technical level.

(ii) Becomes invalid when there is time constraint in the projec schedule or when the users cannot accept the phased deliverables.

Q.34. Explain spiral model in detail and under what circumstances i it beneficial. ***(R.G.P.V., June 201***

Or

Explain how a software development effort is initiated and finall terminated in the spiral model. ***(R.G.P.V., June 201***

Or

"Traditional software process models do not deal sufficiently with th uncertainty". Which model and how solve this problem ? (R.G.P.V., Dec. 201

Ans. The problem with traditional software process models is that the do not deal sufficiently with the uncertainty, which is inherent to softwar projects. Important software project have failed because project risks wer neglected and nobody was prepared when something unforseen happene Boehm proposed a recent model for software development process known a the ***spiral model.*** All the activities in this model can be organized as a spir which has many cycles, as shown in fig. 1.12.

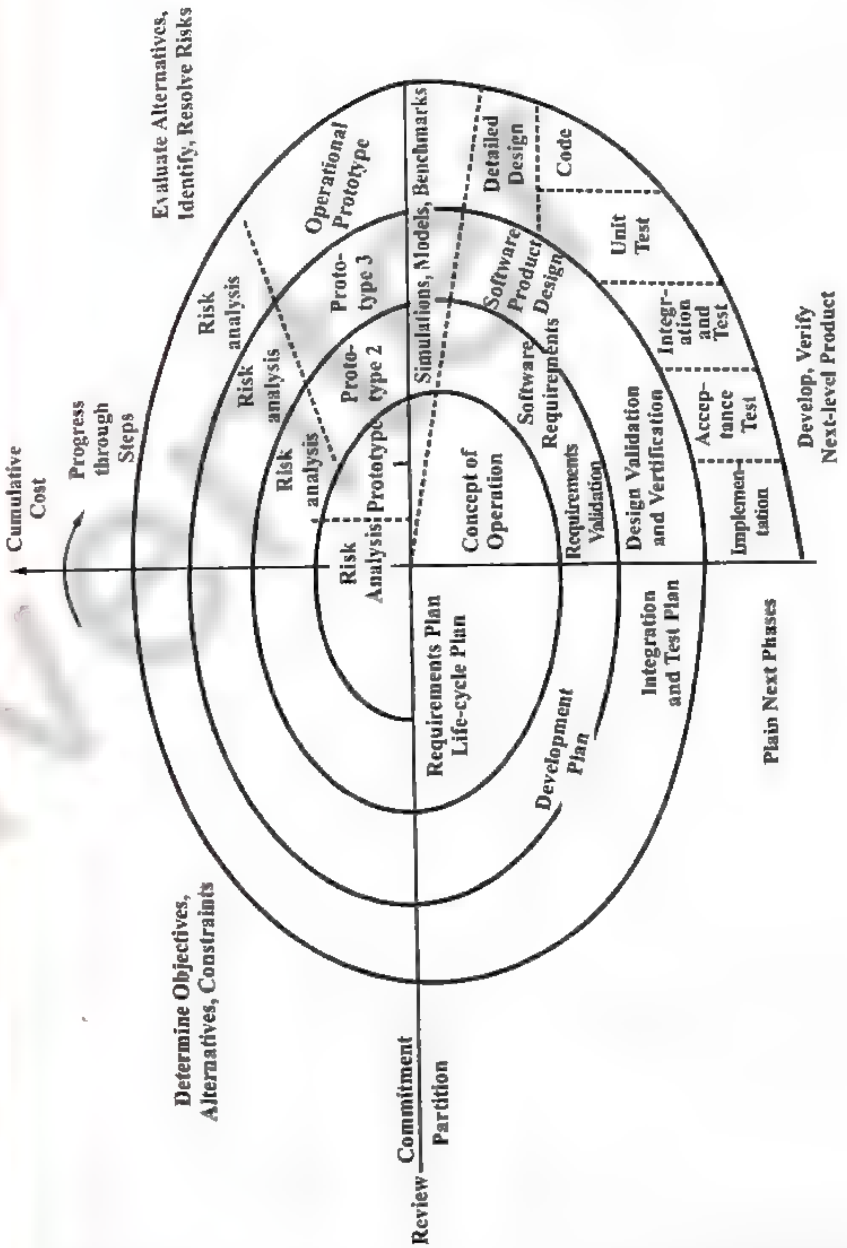


Fig. 1.12 The Spiral Model

In this model, the radial dimension represents the cumulative cost incurred in accomplishing the steps done so far, and the angular dimension represents the progress made in completing each cycle of the spiral. Each cycle in this model begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the objective and constraints that are exist. It is the upper left or first quadrant of the cycle. In the cycle, th next step is to evaluate these different alternatives based on the objectives an constraints. In this step, focus of evaluation is based on the risk perceptio for the project. Risks reflect, the chances that some of the objectives of th project may not be met. To develop strategies is the next step that resolve th uncertainties and risks and it may involve activities such as benchmarkin simulation and prototyping. Next, the software is developed, keeping in min the risks. Then finally the next stage is planned.

The risk-driven nature of the spiral model allows it to accommodate an mixture of a specification-oriented, prototype-oriented, simulation-oriente or some other type of approach. The most important feature of the model i that each cycle of the spiral is completed by a review that covers all th products developed during that cycle including plans for the next cycle. Th spiral model works for development as well as enhancement projects.

A realistic approach to the development of large-scale systems an software is the spiral model. The developer and customer better understan and react to risks at each evolutionary level because software evolves as th process progresses. The prototyping is used as a risk reduction mechanis by the spiral model but, more important, enables the developer to apply th prototyping approach at any stage in the evolution of the product. It maintai the systematic stepwise approach proposed by the classic life cycle bu incorporates it into an iterative framework that more realistically reflects th real world. The spiral model needs a direct consideration of technical risks a all stages of the project and, if properly applied, should minimize risks befor they become problematic.

The spiral model would be the most appropriate development model t follow for projects having many unknown risk that might show up as th development proceeds. For high risk projects this might be a preferred mode

Q.35. Explain software life cycle of spiral model and discuss variou activities in each phase. ***(R.G.P.V., May 201***

Ans. Refer to Q.34.

A spiral model is divided into many framework activities. Typically, ther are between three and six task regions as given below –

(i) Customer Communication – Tasks needed to establish effectiv communication among customer and developer.

(ii) Planning – It is needed to describe resources, timelines an other information.

(iii) Risk Analysis – Tasks required to access both technical an management risks.

(iv) Engineering – It needed to build one or more representations of the applications.

(v) Construction and Release – It needed to construct, test, install and give user support.

(vi) Customer Evaluation – It needed to get customer feedback based on evaluation of the software representations created during engineering stage and implemented during the installation stage.

Q.36. Explain the various stages of WINWIN spiral model. ***(R.G.P.V., June 2017)***

Ans. The WINWIN spiral model is based on the management theory 'Theory W' and approach, i.e. "The necessary and sufficient condition for project success is making winners of all the system's key holder".

The best negotiating skill for a "win-win" result is that when customer gets any product or system that satisfies maximum customer's need then the customer wins and when developer makes realistic and achievable budget in the given deadline then the developer wins.

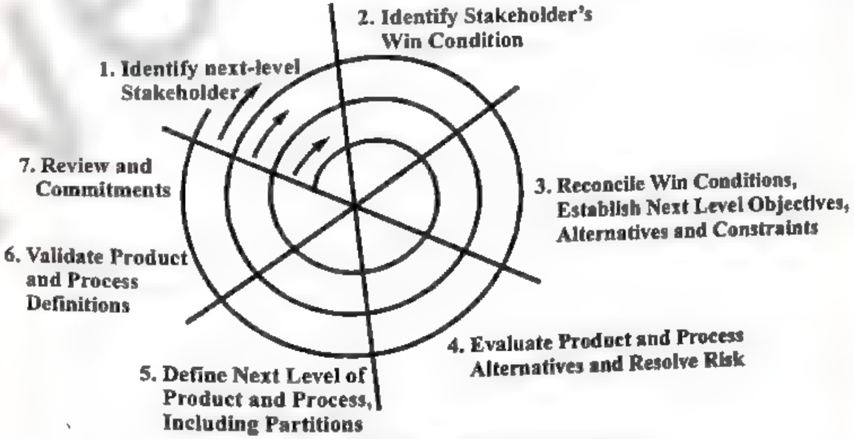

2. Identify Stakeholder's Win Condition
1. Identify next-level Stakeholder
7. Review and Commitments
3. Reconcile Win Conditions, Establish Next Level Objectives, Alternatives and Constraints
6. Validate Product and Process Definitions
4. Evaluate Product and Process Alternatives and Resolve Risk
5. Define Next Level of Product and Process, Including Partitions

Fig. 1.13 WINWIN Spiral Model

There are seven stages of WINWIN spiral model as shown in fig. 1.13.

(i) Identify Next-level Stakeholder – Identify the people or customer whom you want to make happy after successful completion of currently executing phase.

(ii) Identify Stakeholder's Win Condition – Identify the need of the people or customer and document all the goals and objectives.

(iii) Reconcile Win Conditions, Establish Next Level Objectives, Alternatives and Constraints – In this stage, check all the objective, requirements and win conditions are valid and can be achievable within the limited budget.

(iv) Evaluate Product and Process Alternatives, Resolve Risks – this stage, examine the current project carefully to obtain any alternative soluti for problem domain and also resolve risks, if any.

(v) Define Next Level of Product and Process, Including Partition In this stage, next level of product is defined and may partition the produ problem into multiple subproblems that can be developed partially.

(vi) Validate Product and Process Definitions – In this sta product validation is done by checking its various factors.

(vii) Review and Commitments – In this stage, review of all curr completed work is done.

Q.37. Write the advantages and disadvantages of the spiral model.

Ans. There are following advantages and disadvantages of the spiral mode

Advantages –

(i) Avoids the problems resulting in risk-driven approach in the softwa

(ii) Specifies a mechanism for software quality assurance activiti

(iii) Is utilized by complex and dynamic projects.

(iv) Re-evaluation after each step allows changes in us perspectives, technology advances or financial perspectives.

(v) Estimation of budget and schedule gets realistic as the wo progresses.

Disadvantages –

(i) Assessment of project risks and its resolution is not an easy tas

(ii) Difficult to estimate budget and schedule in the beginning, some of the analysis is not done until the design of the software is develope

Q.38. Explain the working of spiral model. Why spiral model considered to be a meta model ? ***(R.G.P.V., June 201***

Ans. **Spiral Model** – Refer to Q.34.

Spiral Model as a Meta Model – Spiral model is considered to be a me model as it incorporates the salient features of both the linear sequential approa as well as of the evolutionary approach. It facilitates the discipline guarante by the sequential approach at the same time it also provides ample flexibility return to the previous stages in a due course of time to address the remaini or previously undiscovered issues. Further, it facilitates both the custom and the developer by tuning their mentality and inculcates effecti communication between them which is esscntial for a timely quality produ It also reduces the risk and facilitates in requirement elicitation and also reduc the product specification anomalies. Spiral model provides greater custom satisfaction and also reduces the maintenance overhead for the developer results in long lasting software products.

Q.39. As you move outward along the process flow path of spiral model what you can say about software that is being developed or maintained ? ***(R.G.P.V., Dec. 2005, 2014)***

Or

As you move outward along the process flow path of the spiral model. What can you say about the software that is being developed ? For what type of projects spiral model is more suitable ? ***(R.G.P.V., June 2007)***

Ans. As the evolutionary process of spiral model begins, the software engineering team moves around the spiral in a clockwise direction, beginning at the center. The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from customer evaluation. In addition, the project manager adjusts the planned number of iterations required to complete the software.

Unlike classical process models that end when software is delivered, the spiral model can be adopted to apply throughout the life of the computer software.

As work moves outward on the spiral, the product moves toward a more complete state and the level of abstraction at which work is performed is reduced (i.e., implementation specific work accelerates as we move further from the origin).

Type of Projects Spiral Model is Suitable – This is a relatively new model; it can encompass different development strategies. In addition to the development activities, it incorporates some of the management and planning activities into the model. For high-risk projects this might be a preferred model.

Q.40. Explain iterative and spiral model of software development. ***(R.G.P.V., Dec. 2016)***

Ans. **Iterative Model** – An iterative life cycle model begins by specifying and implementing just part of the software which can then be reviewed in order to identify further requirements. This process is then repeated producing a new version of the software for each cycle of the model. Following are the four phases of an iterative life cycle model –

(i) Requirement – In this phase, the requirements for the software are gathered and analysed. Iteration should eventually result in a requirements phase that produces a complete and final specification of requirements.

(ii) Design – In this phase, a software solution to meet the requirements is designed. This may be a new design, or an extension of an earlier design.

(iii) Implementation and Test – In this phase, the software is coded, integrated and tested.

(iv) Review – In this phase, the software is evaluated, the curr requirements are reviewed, and changes and additions to requirements propo

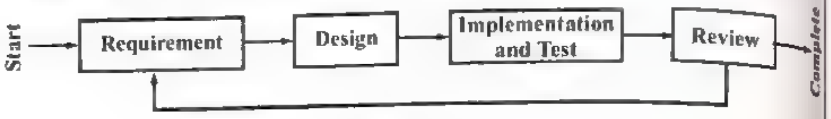


Fig. 1.14

Spiral Model – Refer to Q.34.

Q.41. Compare waterfall and spiral model of software developm How does 'project risk' factor affect the spiral model of software developme (R.G.P.V., Nov./Dec. 20

Ans. **Comparison between Waterfall and Spiral Models** – At first gla the spiral model appears to be the most radical departure from the water model. This is because of the way it is diagrammed.

If the processes within the four cycles of the spiral model were draw a cascading fashion like the waterfall model, the standard processes fr requirements and feasibility through operation and maintenance would fal the same order and would be done is sequence, just as in the waterfall mod

The difference would be in the insertion of the non-standard proces of determining objects, alternatives and constraints evaluating alternativ identifying and resolving risks and planning each cycle.

The nonstandard processes may seem like a trivial difference, but diagrammed, they account for approximately three-quarters of the project eff However, this does not translate to increasing the project time, costs etc. to times what it would take using the waterfall lifecycle. Rather it represent reapportioning of project resources across additional processes that w dramatically reduce the amount of effort required on the standard processe

Effect of Project Risk Factor on Spiral Model – Each cycle of t spiral begins with the identification of the objectives of the portion of prod being elaborated. Next step is to evaluate the alternatives relative to the objecti and constraints.

Frequently this process will identify areas of uncertainity that are signific sources of project risk. If so, the next step would involve the formulation o cost-effective strategy for resolving sources of risk. This may invo prototyping, simulation, bench marking, reference checking and other r resolution techniques.

Once the risks are evaluated, the next step is determined by relat remaining risks. If performance or user-interface risks strongly domina program development or internal interface-control risks, the next step may an evolutionary development one.

Q.42. Discuss component-assembly model and compare it to spiral model. In these models are phases generic. (R.G.P.V., June 2006)

Ans. For a component-based model for software engineering object-oriented technologies provide the technical framework. As shown in fig. 1.15 component assembly model incorporates features of spiral model. It is evolutionary in nature, requiring an iterative approach to the creation of software. However, the component-based development model makes applications from prepackaged software components called classes.

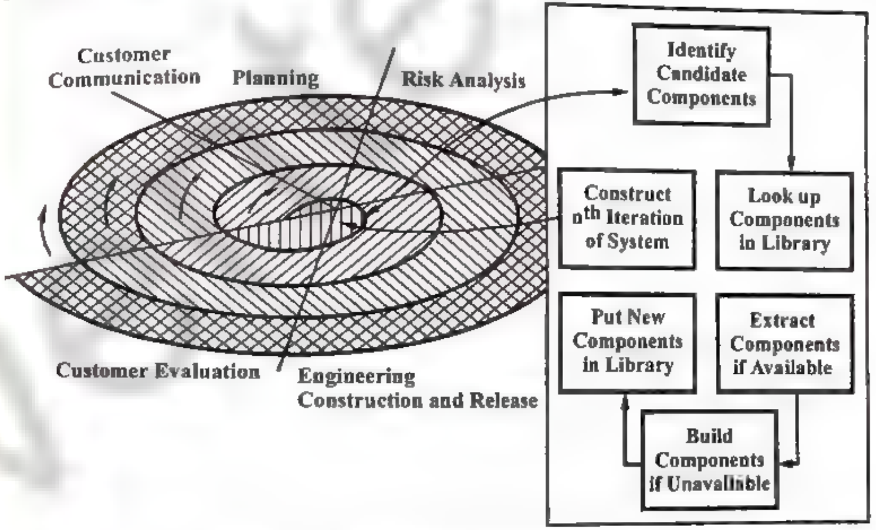

Customer Communication
Planning
Risk Analysis
Customer Evaluation
Engineering Construction and Release
Identify Candidate Components
Construct nth Iteration of System
Look up Components in Library
Put New Components in Library
Extract Components if Available
Build Components if Unavailable

Fig. 1.15 Component-assembly Model

The engineering activity starts with identification of component classes. Once candidate classes are identified, the class library is searched to determine if these classes already exists if they do, they are extracted from the library and reused else it is engineered using object-oriented methods. The first iteration is built and if new is added to library, process goes on till nth iteration.

The component-based development model does the software reuse which indirectly provides software engineers with several measurable advantages. The studies of reusability report component assembly which leads to a 70 percent minimized development cycle time, 84 percent minimized project cost, and a productivity index of 26.2 (comparable with 16.9 which is an industry norm).

Q.43. Describe the evolutionary process model and the model that make use of existing component with example. (R.G.P.V., Dec. 2016)

Ans. Refer to Q.32 and Q.42.

Q.44. What do you mean by software process ? Write a brief note on component-assembly model. (R.G.P.V., June 2014)

Ans. Refer to Q.6 and Q.42.

Q.45. Explain the advantages and disadvantages of the following software process models –

(i) Spiral model (ii) Waterfall model

(iii) Prototype model (iv) Incremental model.

Give at least two example projects where these models can be used for software development. ***(R.G.P.V., Dec. 2009)***

Ans. (i) Spiral Model – Refer to Q.37.

(ii) Waterfall Model – Refer to Q.18.

(iii) Prototype Model – Refer to Q.25.

(iv) Incremental Model – Refer to Q.33.

Types of Projects for which Spiral Model is Suitable – The spiral model is suitable for development of technically challenging and large software products that are prove to several kinds of risks that are difficult to anticipate at the start of the project.

Types of Projects for which Waterfall Model is Suitable – Waterfall model is best suited for projects where requirements are known, stable, and understood.

Types of Projects for which Prototyping Model is Suitable – The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood, however all the risks can be identified before the projects starts.

Types of Projects for which Incremental Model is Suitable – This model is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also used widely for object-oriented development projects.

Q.46. Discuss the RUP process.

Ans. A software development process from rotational is called "RUP". The full form of RUP is "Rational Unified Process". It is a division of "IBM". It divides the development process into four stage that each involve business modeling, analysis and design, implementation, testing, and deployment.

The four stage of RUP process are as follows –

(i) Inception – In this stage we thing about time, resource and labour; Idea for the project is started. The project team determines if the project is worth pursuing and what resources will be needed.

(ii) Elaboration – In this stage the developers consider possible application of the software and costs related with the development. The project's architecture and required resources are further evaluated.

(iii) Construction – In this stage software is written, designed and tested, i.e., the project is fully completed.

(iv) Transition – In this stage software is launched publically for testing. Final adjustments or opdates are made based on feedback from end users.

The RUP development methodology provides a structured way for companies to envision create software programs. Since it provides a specific plan for each step of the development process, it helps prevent resources from being wasted and reduces unexpected developments cost.

Q.47. Describe the agile process. ***(R.G.P.V., Dec. 2016)***

Ans. Agile software development methodology is an process for developing software. However, agile methodology differs significantly from other methodologies. Agile means ability to move quickly and easily and responding swiftly to change. In traditional software development methologies like waterfall model, a project can take several months or years to complete and the customer may not get to see the end product until the completion of the project. At a high level, non-agile projects allocate extensive periods of time for requirements gathering, design, development, testing and user acceptance testing, before finally deploying the project. However, agile process has iterations which are shorter in duration during which pre-determined features are developed and delivered. Agile process has one or more iterations and delivers the complete product at the end of the final iteration.

In agile methodology the delivery of software is unremitting and attention is paid to the good design of the product. In this methodology, the daily interactions are required between the business people and the developers. Changes in the requirements are accepted even in the later stages of the development. However, in agile methodology, the documentation is less. Sometimes, in this methodology, the requirement is not very clear hence its difficult to predict the expected result. The projects following the agile methodology may have to face some unknown risks which can affect the development of the project.

Q.48. Explain software process customization and improvement.

Ans. Refer to Q.57.

Q.49. What is CMM ? Discuss its various levels. ***(R.G.P.V., June 2011)***

Or

Describe about CMM. ***(R.G.P.V., Dec. 2014)***

Or

What is capability maturity model ? ***(R.G.P.V., June 2016)***

Or

Explain the process maturity levels in SEI's CMM. ***(R.G.P.V., June 2017)***

Ans. A customer might feel more confident, for instance if they know that their software supplier is using structured methods. In the United States an influential ***capability maturity model (CMM)*** has been developed at the software engineering institute (SEI), a part of the Carnegie Mellon University. This attempts to place organizations producing software at one of five levels of process maturity which indicate the sophistication and quality of their software production practices. These levels are defined as under –

(i) Level 1 : Initial – The procedures followed tend to be haphazard. Some projects might be successful, but this tends to be because of the skills of particular individuals including project managers. There is no level 0 and so any organization would be at this level by default.

(ii) Level 2 : Repeatable – Organizations at this level will have basic project management procedures in place. However, the way individual tasks are carried out will depend largely on the person doing it

(iii) Level 3 : Defined – The organization has defined the way that each task in the software development lifecycle should be done.

(iv) Level 4 : Managed – The products and processes involved in software development are subject to measurement and control.

(v) Level 5 : Optimizing – Improvement in procedures can be designed and implemented using the data gathered from the measurement process.

PRODUCT AND PROCESS METRICS

Q.50. What are measures, metrics and indicators ? (R.G.P.V., June 2006)

Ans. Within the context of software engineering, a "measure" provides a quantitative indication of the amount, dimension, extent, capacity, or size of some attributes of a product or process. While, the IEEE Standard Glossary of Software Engineering Terms, defines "metric" as a quantitative measure of the degree upto which a system, component, or process have a given attribute.

When a single data point has been stored, a measure has been established. A software metric relates the individual measures in some specified way. A software developer collects measures and develops metrics to get "indicators" which is a single metric of combination of metrics providing the knowledge of what happens inside the software process, project, or product itself. An indicator provides a way that enables the project manager or software engineer to adjust the process, project, or the process of providing the things.

For instance, there are four software teams which are working on a large software project. Each team will have to conduct design reviews but is limited to choose the type of review that it will use. After examining metric, errors discovered per person-hour expended, the project manager points that the two teams using more formal review techniques have errors encountered per person-hour expended that is 40% higher than the other teams. Considering all parameters equal, this gives an indicator that formal review methods may provide a higher return on time investment than another, less formal review approach. The project manager may then decide to suggest that all teams use the more formal approach. The metric gives the project manager with insight which results in informed decision making.

Q.51. Explain the term project indicator. (R.G.P.V., Dec. 2005)

Ans. ***Project indicators*** allow a software project manager for doing the following tasks –

(i) To assess the status of the currently running project

(ii) To track potential risks

(iii) To show problem areas before being critical

(iv) To adjust work flow or tasks

(v) To evaluate the performance of the project team in quality controlling of software work products.

Q.52. Discuss briefly about product metrics.

Ans. Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Products metrics are concerned with characteristics of the software itself. Product metrics fall into two classes –

(i) Dynamic metrics which are collected by measurements made of a program in execution.

(ii) Static metrics which are collected by measurements made of the system representations such as the design, program or documentation.

Dynamic metrics help to assess the efficiency and the reliability of a program whereas static metrics help to assess the complexity, understandability, and maintainability of a software system.

Dynamic metrics are usually fairly closely related to software quality attributes. It is relatively easy to measure the execution time required for particular functions and to assess the time required to start up a system. These relate directly to the system's efficiency. Similarly, the number of system failures and the type of failure can be logged and related directly to the reliability of the software.

On the other hand, static metrics have an indirect relationship with quality attributes. A large number of these metrics have been proposed and experiments carried out to try to derive and validate the relationships between these metrics and system complexity, understandability, and maintainability. Several static metrics which have been used for assessing quality attributes, given in table 1.1. Of these, program or component length and control complexity seem to be the most reliable predictors of understandability, system complexity and maintainability.

Table 1.1 Software Product Metrics

S.No.	*Software Metric*	*Description*
(i)	Fan-in/Fan-out	Fan-in is a measure of the number of functions that call some other function (say X). Fan-out is the number of functions which are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
(ii)	Length of code	This is a measure of the size of a program. Generally the larger the size of the code of a program component, the more complex and error-prone that component is likely to be.
(iii)	Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability.
(iv)	Length of identifiers	This is a measure of the average length of distinct identifier in a program. The longer the identifiers the more likely they are to be meaningful and hence the more understandable the program.
(v)	Depth of conditional nesting	This is a measure of the depth of nesting of if statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
(vi)	Fog index	This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document may be to understand.

Q.53. Explain the concept of software measurement. What is the difference between direct and indirect measures ?

Or

Write short note on software measurement.

(R.G.P.V., June 2005, Dec. 200

Ans. Software measurement is considered as a management tool which, if conducted properly, helps the project manager and the entire software teams to take decisions for successful completion of the project. Measurement process is characterized by a set of five activities these are as follows –

(i) Formulation – This performs measurement and develops suitable metrics for software under consideration.

(ii) Collection – This gathers data to derive the formulated metrics.

(iii) Analysis – It computes metrics and the use of mathematical tools.

(iv) Interpretation – This analyzes the metrics to achieve insight into the quality of representation.

(v) Feedback – This communicates recommendation derived from product metrics to the software team.

Measurements are classified in – direct measures (e.g., the length of a bolt) and indirect measures (e.g. the quality of bolts produced, measured by counting rejects). Similarly, software metrics are classified in direct and indirect measures. Direct measures of the software engineering process are cost and effort expended.

Direct measures of the product are lines of code (LOC) generated, execution speed, memory size, and defects discovered over some set time period. In contrast, indirect measures of the product discovered quality, complexity, functionality, reliability, efficiency, maintainability, and so on.

As long as specific conventions for measurement are determined in advance, the direct measures like the cost and effort required for software construction, the number of code lines produced, and many more, are relatively easy to collect. However, the software quality and functionality or even its efficiency or maintainability are more complex to assess and can only be measured indirectly.

Q.54. What do you mean by software complexity ? How complexity of software is measured ? (R.G.P.V., June 2015)

Ans. Software complexity is usually analyzed as a static (i.e., compile-time) property of source code, not an execution-time property. It has the most direct bearing on the extent of required software testing, but also, it is an indicator of difficulty in software maintenance, particularly program comprehension. As software complexity increases, development effort also increases, although this is a little deceptive since much of the analysis is based on existing code. Finally, an awareness of software complexity may lead to improved programming practices, and even better design techniques.

Measurement of Software Complexity – Programmers feel it difficu to gauge the code complexity of an application, which makes the concep hard to understand. Two common code complexity measures are the McCab metric and Halstead's software science. The McCabe metric determines cod complexity on the basis of the number of control paths created by the cod While this information provides only a portion of the complex picture, it give an easy-to-compute, high-level measure of program's complexity. The McCab metric is used for testing. Halstead approach is based on the mathematic relationships among the number of variables, the complexity of the code an the type of programming language statements. However, Halstead's work i criticized for its complicated computations as well as questionable methodolog for acquiring some mathematical relationships.

McCabe proposed a method that maps a program to a directed, connecte graph. In the graph, nodes represent decision or control statements. Th directions specify control paths that dictate the program flow. The enclose regions in the graph represent code chunks that execute on basis of the decisio or control statements.

McCabe said that the complexity of a program equals the number o enclosed regions in its mapped graph plus one. This number is called as th cyclomatic complexity of the program.

McCabe calculated the cyclomatic complexity of a group of programs By comparing the cyclomatic complexity with the frequency of errors, h found that a program with cyclomatic complexity more than 10 is error pron so he said that programs should not have cyclomatic complexity more than 10

Q.55. Explain the term process indicator. ***(R.G.P.V., Dec. 2005***

Ans. Process indicators allow a software engineering organization to g insight into the efficacy of the current process. Also, they allow managers an practitioners to determine what works and what doesn't. Process metrics ar gathered over all projects and over a long time period, so as to provide indicato leading to long-term software process improvement.

Q.56. Explain what are process metrics. Differentiate between produ and process metrics. ***(R.G.P.V., Dec. 201***

Ans. Process measurements are quantitative data for the software proces Humphrey suggests that the collection of process metrics is necessary fo process improvement. Humphrey also argues that measurement has a significa role to play in small-scale, personal process improvement. Process metrics, i their own right, are used to assess whether or not the efficiency of a proces has been enhanced. Effective improvements to the testing process shou

minimize the effort, testing time or both. However, process measurements on their own cannot be used to determine if product quality has enhanced. Product metrics must also be gathered and related to the process activities.

There are three classes of process metric that can be collected –

(i) The Time Taken for a Specific Process to Completed – This is the total time spent on process, calendar time, the time spent on the process by particular engineers, etc.

(ii) The Resources Needed for a Specific Process – The resources might be total effort in person-days, computer resources, travel costs etc.

(iii) The Number of Occurrences of a Specific Event – Examples of events that might be monitored are the average number of lines of code modified in response to a requirements change, the number of defects found during code inspection, the number of requirements changes requested etc.

The first two types of measurement are used to assist discover if process changes has enhanced the efficiency of a process. There are fixed points in a software development process like acceptance of requirements, the completion of architectural design, the completion of test data generation etc. It may be possible to measure the time and effort needed to move from one of these fixed points to another. The measured values may be used to suggest areas where the process might be enhanced. After changes have been produced, measurements of system attributes can show if the process changes have actually been advantageous.

Measurements of the number of events which take place can have a more direct bearing on software quality. For instance, increasing the number of defects found by changing the program inspection process will probably be reflected in improved product quality.

Difference – Product metrics help to measure the characteristics of a product being developed. LOC (lines of code) and function point to measure size, PM (person-month) to measure the effort required to develop it, time complexity of the algorithm, months to measure required time for developing the product, etc. are all examples of product metrics. On the other hand, process metrics help to measure how a process is performing. Examples of process metrics are average number of defects found per hour of inspection, average defect correction time, average number of failures detected during testing per LOC, review effectiveness, number of latent defects per line of code in the developed product, productivity.

Q.57. How to make any software process effective ? How to measure the effectiveness ?

Or

Explain process metrics and software process improvement.

(R.G.P.V., June 2005)

Ans. There is only one rational way to improve any software process i.e., to measure specific process attributes, develop a collection of meaningful metrics on the basis of these attributes, and then use the metrics to obtain indicators leading to a strategy for improvement.

It is important to note that process is only one of a number of controllable factors in improving software quality and system performance. Fig. 1.16 shows the position of the process in a software environment.

Observe the fig. 1.16, process is at the centre of a triangle connecting three factors influencing the software quality and system performance. The skill and motivation of people has been depicted to be the single most influential factor. While the product complexity also has a substantial impact on quality and performance. And on the third end, technology that populates the process also has an impact. This process triangle exists under a circle of environmental conditions including the development environment, business conditions, and customer characteristics.

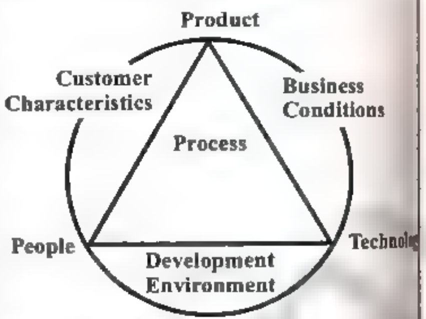


Fig. 1.16 Determinants for Software Quality and Organizational Effectiveness

Now the effectiveness of a software process can be measured indirectly i.e., we derive a set of metrics on the basis of the results derived for the process including measures of errors uncovered before software release, defects provided to and encountered by the customers, work products delivered i.e., productivity human effort spent, calendar time spent, schedule conformance, and several other measures. Also, process metrics are derived by measuring the characteristics of particular software engineering tasks.

Grady states that there are private and public uses for different types of process data. Since it is natural that individual software engineers might be sensitive to the use of metrics collected on an individual basis, these data should be private to the individual and work as an indicator for the individual only. Examples of *private metrics* are defect rates (by individual), defect rates (by module), and errors discovered during development.

The private process data philosophy conforms well with the personal software process approach suggested by Humphrey. Humphrey identifies that software process improvement can and should start at the individual level. Private process data can serve as an important driver as the individual software engineer works to improve.

For the software project team, some private metrics are private but public to all team members. Examples are defects reported for major software functions, errors found during formal technical reviews, and lines of code or function points per module and function. These data are reviewed by the team to uncover indicators that can enhance team performance. Public metrics assimilate information that was private to individuals and teams. Project level defect rates, effort, calendar times, and related data are collected and evaluated in an attempt to uncover indicators that can enhance organizational process performance.

Q.58. Define SSPI and explain the failure analysis used by SSPI.

Or

Explain statistical software process improvement. (R.G.P.V., Dec. 2005)

Ans. Statistical software process improvement (SSPI) is a rigorous approach obtained by the derivation of simple indicators. In short, SSPI uses software failure analysis to get information about all errors and defects found as an application, system, or product is developed and used. Failure analysis steps are as follows –

(i) All errors and defects are classified by origin.

(ii) The cost of correcting each error and defect is noted.

(iii) The number of errors and defects in each class is counted and placed in ascending order.

(iv) The total cost of errors and defects in each class is calculated.

(v) Results are analyzed to get the class having highest cost of the organization.

(vi) Methods are discovered to modify the process, which eliminates the class of errors and defects which is most costly.

Fig. 1.17 illustrates a simple defect distribution for four software projects, which can be developed by following steps (i) and (ii), noted above. Here, eight causes of defects and their origins are shown. "Grady" recommends that the development of "fishbone diagram" (shown in fig. 1.18) to assist in diagnosing the data represented in the frequency diagram.

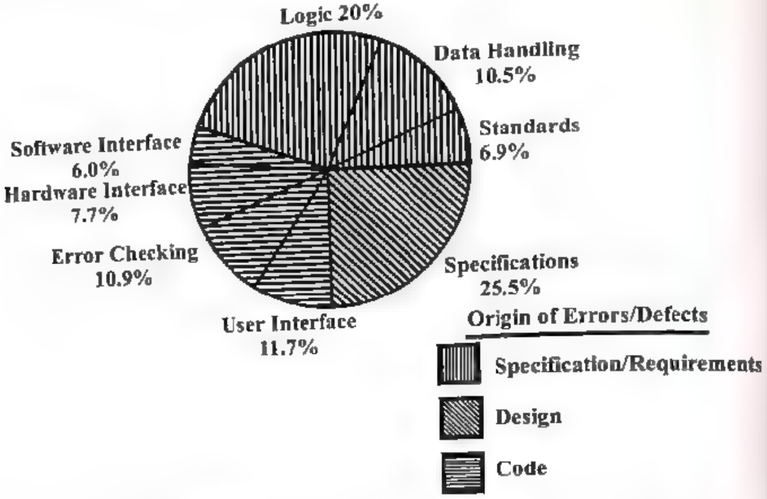


Fig. 1.17 Causes of Defects and their Origins for Four Software Projects

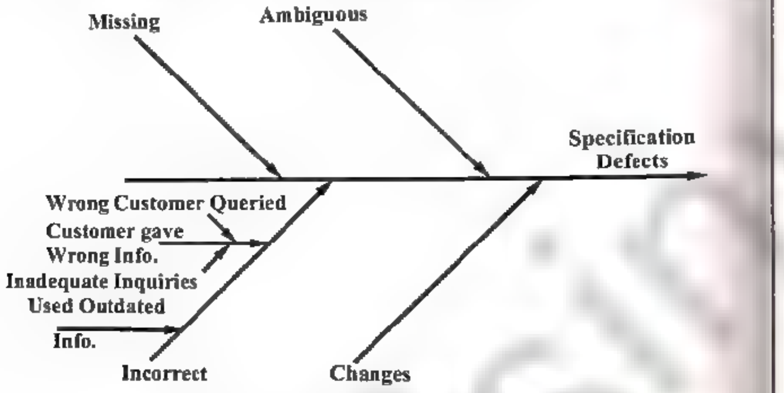


Fig. 1.18 A Fishbone Diagram

Observe the fig. 1.18, the spine of the diagram i.e., central line represents the quality factor under consideration i.e., specific defects accounting 25% of the total. Each of the four ribs i.e., diagonal lines connecting to the spine represent potential causes for the quality problem i.e., missing requirements, ambiguous specifications, incorrect requirements, changed requirements. The spine and ribs notation are then appended to each of the major ribs to expand upon the cause noted (here, expansion is depicted only for the incorrect cause).

Q.59. What are software metrics ? What is the role of metrics in process management ?

(R.G.P.V., Dec. 2008)

Ans. The IEEE standard Glossary of Software Engineering Terms defines ***metric*** as "a quantitative measure of the degree to which a system, component or process possesses a given attribute."

Software metrics are quantifiable measures that could be used to measure different characteristics of a software system or the software development process. All engineering disciplines have metrics to quantify various characteristics of their products. Software metrics is an emerging era. Because the software has no physical attributes, conventional metrics are not much helpful in designing metrics for software. A number of metrics have been proposed to quantify things like the size, complexity, and reliability of a software product, metrics provide the scale for quantifying qualities; actual measurement must be performed on a given software system in order to use metrics for quantifying characteristics of the given software.

Values of some metrics can be directly measured while others might have to be deduced by other measurement, such metrics are called indirect metrics.

Role of Metrics in Process Management – Process metrics are gathered across all projects and over long periods of time. Their purpose is to provide indicators that results in long-term software process improvement. The only rational way to improve any process is to measure certain attributes of the process, develop a set of meaningful metrics on the basis of these attributes and then use the metrics to give indicators that will result in a strategy for improvement because the process is only one of a number of "controllable factors in improving software quality and organizational prformance."

The efficacy of a software process is measured indirectly. It means that, a set of metrics is derived on the basis of the outcomes that can be derived from the process. Outcomes are measures of errors uncovered before release of the software, defects delivered to and reported by end-users, work products delivered human effort extended, time expended, etc.

Software process metrics can provide significant benefit as an organization works to improve its overall level of process maturity.

Q.60. What are software metrics ? What are the benefits of using software metrics ? How do software metrics improve the software process ?

(R.G.P.V., June 2010)

Ans. **Software Metrics** – Refer to Q.59.

Benefits of Software Metrics –

(i) Macro-estimating

(ii) Evaluating bids

(iii) Evaluating bids in-house

(iv) Comparing an estimate with part projects

(v) Micro-estimating

(vi) Allocating resources

(vii) Predicting code completion

(viii) Predicting defects per month

(ix) Assessing progress

(x) Assessing process improvement

(xi) Assessing project improvement

(xii) Project replanning.

Software Metrics and Software Process Improvement – Refer to Q.57.

Q.61. Show why and how software metrics can improve the software process. Enumerate the effects of metrics on software productivity.
(R.G.P.V., Nov./Dec. 2007, Dec. 2008, June 2015)

Ans. **Software Metrics and Software Process Improvement** – Refer to Q.57.

Effects of Metrics on Software Productivity – Metrics, by initiating observation, have led organizations (and individuals) to a process of self-discovery of goals, capabilities, and constraints. Quantitative expressions of the observations have brought in additional clarity and simplicity.

Metrics are seen as force multipliers in improvement initiatives and quality movements. On the one hand, the ability to improve is aided by the ability to measure (to see). By integrating knowledge and providing better communication, resources are better utilized and efforts are better rewarded.

Structured thinking, a prerequisite for metrics, has paved the way for systems creation in unexpected areas. For instance, inspired by metrics data patterns, estimation models for bug fixing have been constructed and as a sequel the bug estimation task has been refined and redefined in many organizations.

Metrics data fills in human brains, gradually and almost imperceptibly. Over time, the personal thinking process gets enriched with fresh data and fresh learning. Beyond rational models, metrics also lead to cognitive intuitive models, perfecting a skill that comes naturally to human beings vision.

The most celebrated contribution of metrics is the decision-making support it provides. The first revolution metrics created is the information revolution. Over a period of time, information support has changed its style and moved from Management Information System (MIS) to Decision Support System (DSS), through the well-known phases of evolution.

Progress in metrics data analysis has created new and economic ways of creating knowledge assets in organizations. In that sense, metrics is a rudimentary knowledge engine Constant interpretations of metrics inject a stream of values into the organization, some temporary, many more enduring. The learning process being what it is, experimental values and tentative knowledge structures all become part of the global repository of knowledge assets.

Problem-solving cycles have benefited from metrics in all the phases. All scales of measurement are useful here. Metrics are used for recognition and later for diagnostics of problems. Experiments are conducted of test ideas, true to the scientific spirit of metrics application.

Q.62. Describe two metrics which are used to measure the software in detail. ***(R.G.P.V., May 2019)***

Ans. The two metrics which are used to measure the software are as follows –

(i) Size-oriented Software Metrics – Size-oriented software metrics are derived considering the size of the software produced, since size is the major factor that affects the cost of a project. Size in itself is of little use, it is the relation of size with the cost and quality that makes size an important metric. At the end of the project, size is measured primarily to record it, alongwith the total cost, for future references. Table 1.2 illustrates such size-oriented measures.

Table 1.2 Size-oriented Metrics

Project	*Lines of Code*	*Effort*	*$(000)*	*Pp.doc.*	*Errors*	*Defects*	*People*
Alpha	12,200	24	168	365	135	30	3
Beta	27,400	62	442	1224	322	88	5
Gamma	20,100	43	315	1050	258	65	6
Delta	25,000	50	605	1150	155	72	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Observe the table 1.2, it lists each software development project that has been completed over the past few years and corresponding measures for that project. This type of data are used to determine the cost estimation models for a process. It is also used to measure productivity during the project. For example, referring to table 1.2, for project alpha – 12,100 lines of code were developed with 24 person-months of effort at a cost of $168,000. (Note that, effort and cost include all software engineering activities i.e., analysis, design, code, and test, not just coding.) Furthermore, project alpha requires 365 pages of documentation, and 135 errors were found before software was released, and 30 defects were encountered after release to the customer within the first operating year. Moreover, 3 people worked on the software project development.

The most common measure of size is delivered lines of code. From the rudimentary data contained in the table, the following set of simple size-oriented metrics can be developed for each project –

(a) Errors per KLOC (thousand lines of code)
(b) Defects per KLOC
(c) $ per LOC
(d) Page of documentation per KLOC.

Furthermore, other interesting metrics can also be computed as follows –

(e) Errors per person-month
(f) LOC per person-month
(g) $ per page of documentation.

Size-oriented metrics are not the best ways for the software development process measurement. Most of the controversy swirls around the use of lines of code as a key measure. The trouble with LOC is that the number of lines of code for a project depends heavily on the language used. For example, an assembly language program will be large compared to the program written in a higher-level language, if LOC is used as a size measure. Even for the same language, the size can vary considerably depending on the programmer and other factors.

(ii) Function-oriented Software Metrics – Function-oriented software metrics are used as a measure of the functionality delivered by the application as a normalization value. Functionality must be derived indirectly, as it cannot be directly measured. Functionality can be measured by computing function points, which are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

Functions points are computed by completing the table 1.3, given below.

Table 1.3 Computing Function Points

Measurement Parameter	*Count*	×	*Weighting Factor*			
			Simple	*Average*	*Complex*	
Number of user inputs	□	×	3	4	6	= □
Number of user outputs	□	×	4	5	7	= □
Number of user inquiries	□	×	3	4	6	= □
Number of files	□	×	7	10	15	= □
Number of external interfaces	□	×	5	7	10	= □
Count total →						□

Observe the table 1.3, where five information domain characteristics are determined and counts are provided in the appropriate location of the table. Information domain values are defined as follows –

(i) Number of User Inputs – Each user input, providing different application-oriented data to the software, is counted. These should be separated from inquiries, which are counted distinctly.

(ii) Number of User Outputs – Each user output, providing application-oriented information to the user, is counted. Here, output refers to reports, error messages, etc. Individual data items are not counted separately.

(iii) Number of User Inquiries – Inquiry is an on-line input that generates some immediate software response in the form of an on-line output. Here, each distinct inquiry is counted.

(iv) Number of Files – Each logical master file is counted. Logical master file refers to a long grouping of data that may be one part of a large database or a separate file.

(v) Number of External Interfaces – All external, machine readable interfaces that are used to send information to another system are counted.

Once the above noted data have been collected, a complexity value is associated with each count. Function points (FP) can be computed by using the following relationship –

$$FP = \text{Count total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$

where F_i (i = 1 to 14) are complexity adjustment values.

Based on responses to the 14 specific queries or algorithms. The value of F_i may range from 0 to 5 as follows –

Table 1.4

S.No.	*Response*	*Value of F_i*
(i)	Not present	0
(ii)	Insignificant influence	1
(iii)	Moderate influence	2
(iv)	Average influence	3
(v)	Significant influence	4
(vi)	Strong influence	5

Function points, once calculated, are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and the following other attributes –

(i) Errors per FP
(ii) Defects per FP
(iii) $ per FP
(iv) Pages of documentation per FP
(v) FP per person-month.

Q.63. Discuss in brief extended function point metrics.

(R.G.P.V., June 20

Ans. The function point measure was basically designed for the busi information systems applications. To accommodate these applications, data dimension or information domain values were emphasized to the exclu of the functional and behavioural dimensions. Thus, the function point meas was insufficient for many engineering and embedded systems which empha function and control.

A function point extension, referred to as ***feature point,*** is a superse the function point measure used in systems engineering software applicatio It accommodates applications having high algorithmic complexity, for exam real-time, process control, and embedded software applications.

To compute the feature point, information domain values (or d dimension) are again counted and weighted. Also, the feature point me counts a new software characteristic called ***algorithms***, for example, inver a matrix, decoding a bit string, or handling an interrupt.

Another function point extension, for real-time systems and enginee products, integrates the data dimension of software with the functional behavioural dimensions to provide a function-oriented measure called the ***function point***. It provides an indication of the functionality delivered by software. Characteristics of all three software dimensions are "coun quantified, and transformed" into a measure that offers an indication of functionality delivered by the software.

Counts of retained data and external data are used alongwith measure complexity to derive a ***data dimension count.***

The functional dimensional is measured by considering "the numbe internal operations needed to transform input to output data". Due to function point computation, a "transformation" is considered as a sequenc processing steps that are constrained by a set of semantic statements. level of complexity assigned to each transformation is a function of the nu of processing steps and the number of semantic statements that control processing steps. Table 1.5 provides guidelines for assigning complexit the functional dimension.

Table 1.5 Determining the Complexity of a Transformation for 3D-function Points

Semantic Statements / *Processing Steps*	*1-5*	*6-10*	*11+*
1-10	Low	Low	Average
11-20	Low	Average	High
21+	Average	High	High

The control dimension is measured by counting the number of transition between states. A state represents some externally observable mode of behaviour and a transition occurs as a result of some event that causes the software or system to change its mode of behaviour.

The following relationship is used to compute 3D-function points –

$$\text{Index} = I + O + Q + F + E + T + R$$

where I, O, Q, F, E, T, and R denote complexity weighted values for the following elements i.e., inputs, outputs, inquiries, internal data structures, external files, transformation, and transitions, respectively. Each of the complexity weighted values is calculated by using the following relationship –

$$\text{Complexity weighted value} = N_{il}W_{il} + N_{ia}W_{ia} + N_{ih}W_{ih}$$

where N_{il}, N_{ia}, and N_{ih} denote the number of occurrences of element i, for each complexity level i.e., low, medium, high. And, W_{il}, W_{ia}, and W_{ih} are the corresponding weights. Table 1.5 illustrates the overall complexity of a transformation for 3D-function points.

The overall computation for 3D-function point is shown in fig. 1.19.

	Complexity Weighting			
Measurement Element	**Low**	**Average**	**High**	
Internal Data Structures	□ × 7 +	□ × 10 +	□ × 15 =	□
External Data	□ × 5 +	□ × 7 +	□ × 10 =	□
Number of User Inputs	□ × 3 +	□ × 4 +	□ × 6 =	□
Number of User Outputs	□ × 4 +	□ × 5 +	□ × 7 =	□
Number of User Inquiries	□ × 3 +	□ × 4 +	□ × 6 =	□
Transformations	□ × 7 +	□ × 10 +	□ × 15 =	□
Transitions	□ × n/a +	□ × n/a +	□ × n/a =	□
3D-function Point Index	→			□

Fig. 1.19 Computing the 3D-function Point Index

Q.64. What are the relative advantages of using either the LOC or the function point metric to measure the size of a software product ? List important shortcomings of the LOC as a software size metrics. (R.G.P.V., Nov./Dec. 2007)

Ans. **Advantages of LOC –** LOC is an artifact of all software development projects that can be easily counted, that many existing software estimation models use LOC or KLOC as a key input.

Advantages of FP – (i) FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages.

(ii) FP is an attractive estimation approach as it depends on that are more likely to be known early in the evolution of a project.

(iii) Another advantage of function points over DLOC is that definition of DFP depends only on information available from the specifica whereas the size in DLOC cannot be directly determined from specificati

Shortcomings of the LOC – The trouble with LOC is that the num of lines of code for a project depends heavily on the language used. For exam a program written in assembly language will be large compared to the s program written in a high-level language, if LOC is used as a size meas Even for the same language, the size can vary considerably depending on programmer and other factors. What forms a line in determining the si also not universally accepted and depends on the use of the size measure, example, if we are interested in size to determine the total effort, then it m be reasonable to include comment and data lines.

Limitations of FP – (i) FP computation depends on subjective ra than objective data.

(ii) The counts of the information domain are difficult to co after the fact.

(iii) FP has no direct physical meaning. It is just a number.

NUMERICAL PROBLEMS

Prob.1. Compute the function point value for a project with following information domain characteristics –

	Average Weighing Factors
No. of user input = 32	*4*
No. of user output = 60	*5*
No. of user required = 24	*4*
No. of files = 8	*10*
No. of external interfaces = 2	*7*

Assume all complexity adjustment and weighing factors ave Assume that 14 algorithms have been counted. ***(R.G.P.V., Dec. 2***

Or

Compute the function point value for a project with the follo information domain characteristics –

(i) No. of external inputs – 32
(ii) No. of external outputs – 60
(iii) No. of external inquiries – 24
(iv) No. of internal logic files – 08
(v) No. of external interface files – 02

Assume that all complexity adjustment values are average.
(R.G.P.V., June 2008, 2

Sol.

S. No.	*Measurement Parameter*	*Count*	×	*Weighting Factor*			
				Simple	*Ave.*	*Complex*	
(i)	No. of user input	32	×	3	4	6	= 128
(ii)	No. of user output	60	×	4	5	7	= 300
(iii)	No. of user inquiry	24	×	3	4	6	= 96
(iv)	No. of files	8	×	7	10	15	= 80
(v)	No. of external interfaces	2	×	5	7	10	= 14

Count Total ⟶ = **618**

F_i for average case = 3

Sum of all F_i (from 1 to 14) = 14 × 3 = 42

$$\therefore \quad FP = \text{Count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$
$$= 618 \times [0.65 + 0.01 \times 42]$$
$$= 618 \times [0.65 + 0.42]$$
$$= 618 \times 1.07 = \mathbf{661.26}$$

Ans.

For Feature Point –

S. No.	*Measurement Parameter*	*Count*	×	*Weighting Factor*			
				Simple	*Ave.*	*Complex*	
(i)	No. of user input	32	×	3	4	6	= 128
(ii)	No. of user output	60	×	4	5	7	= 300
(iii)	No. of user inquiry	24	×	3	4	6	= 96
(iv)	No. of files	8	×	7	10	15	= 80
(v)	No. of external interfaces	2	×	5	7	10	= 14
(vi)	No. of algorithms	14	×	–	–	–	= 14

Count Total ⟶ = **632**

F_i (complexity adjustment value) for average case = 3

$$\Sigma F_i = 14 \times 3 = 42$$
$$\therefore \quad \text{Feature point} = \text{Count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$
$$= 632 \times [0.65 + 0.01 \times 42]$$
$$= 632 \times [0.65 + 0.42]$$
$$= 632 \times 1.07 = 676.24$$
$$\therefore \quad \text{Feature point} = \mathbf{676.24}$$

Ans.

Note – (i) Weighting factors for average case are used.

(ii) Σ (F_i) are complexity adjustment values for all 14 queries.

(iii) For calculating feature point one more measuring characteri i.e., algorithms are added to the table for calculating unadjusted functi point or feature point.

Prob.2. Consider a project with the following functional unit –

Number of user inputs = 50
Number of user outputs = 40
Number of user enquires = 35
Number of user files = 06
Number of external interfaces = 04.

Assume all complexity adjustment factors and weighting factors averages.

Compute the function points for the project.

(R.G.P.V., Dec. 20

Sol.

S. No.	*Measurement Parameter*	*Count*	×	*Weighting Factor*			
				Simple	*Average*	*Complex*	
(i)	No. of user input	50	×	3	4	6	= 2
(ii)	No. of user output	40	×	4	5	7	= 2
(iii)	No. of user enquires	35	×	3	4	6	= 1
(iv)	No. of files	06	×	7	10	15	= 6
(v)	No. of external interfaces	04	×	5	7	10	= 2

Count Total ⟶ = 6

F_i for average case = 3

Sum of all F_i (from 1 to 14) = 14 × 3 = 42

$\therefore$ Function point = Count total × [0.65 + 0.01 × $\Sigma(F_i)$]

$$= 628 \times [0.65 + 0.01 \times 42]$$
$$= 628 \times [0.65 + 0.42]$$
$$= 628 \times 1.07$$
$$= \mathbf{671.96}$$

UNIT 2

REQUIREMENT ELICITATION, ANALYSIS AND SPECIFICATION

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS, REQUIREMENT SOURCES AND ELICITATION TECHNIQUES

Q.1. What is a software requirement ? What are its objectives ?

Ans. Requirement is a condition or capability possessed by a software or system component in order to solve a real world problem. The problems can be to automate a part of a system, to correct shortcomings of an existing system, to control a device, and so on. IEEE defines requirement as –

(i) A condition or capability needed by a user to solve a problem or achieve an objective.

(ii) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

(iii) A documented representation of a condition or capability as in (i) or (ii).

Objectives – The objectives of software requirements are as follows –

(i) To introduce the concepts of user and system requirements.

(ii) To describe functional and nonfunctional requirements.

(iii) To explain two techniques for describing system requirements.

(iv) To explain how software requirements may be organized in a requirements document.

Q.2. Discuss the role of software requirements ?

Ans. Software requirements serve two major roles in a development effort. They specify what to develop and when the development is completed. A requirements document has all the software requirements of the system that is to be developed. It communicates the customer's needs, wishes, and expectations to the developers of the system. A requirements document may also have descriptions of the required computational functionality and the system behaviour, guidelines for the user interface, technical aspects of the hardw software interface, and operational characteristics.

The second role of software requirements is to form the basis determining when the software product is to be completed. The verifica of the software functionality against the requirements document serve demonstrate that the contractual agreement between the customer and developers has been met and generally implies the end of the developm phase. This verification may need the construction of test situation objectively prove that the software products comply with the requireme document.

Q.3. What are functional and non-functional requirements ?
(R.G.P.V., June 20

Or

Compare the functional and non-functional requirements.
(R.G.P.V., Dec. 20

Ans. **Functional Requirements** – The functional requirements, called ***behavioural requirements***, describe the functionality or services software should provide. For this, functional requirements describe interaction of software with its environment and specify the inputs, outp external interfaces, and the functions that should not be included in the softw Also, the services provided by functional requirements specify the proced by which the software should react to particular inputs or behave in partic situations. IEEE defines function requirements as a function that a system component must be able to perform.

Consider for example the functional requirements of an online bank system –

(i) The user of the bank should be able to search the desired servi from the available ones.

(ii) There should be appropriate documents for users to read. T implies that when a user wants to open an account in the bank, the fo must be available so that the user can open an account.

(iii) After registration, the user should be provided with a uni acknowledgement number so that he can later be given an account numb

These requirements indicate user requirements and specify that functio requirements may be described at different levels of detail in an online bank system.

The functional requirements should be complete and consiste Completeness implies that all the user requirements are defined. Consisten implies that all requirements are specified clearly without any contradictio definition. Generally, it is observed that completeness and consistency ca

be achieved in large software or in a complex system due to the errors that arise while defining the functional requirements of these systems.

Non-functional Requirements – The non-functional requirements, also called ***quality requirements***, relate to system attributes such as reliability and response time. Non-functional requirements arise due to user requirements, budget constraints, organizational policies etc. These requirements are not related directly to any particular function provided by the system.

Non-functional requirements should be accomplished in a software to make it perform efficiently. For example, if an aeroplane is unable to fulfil reliability requirements, it is not approved for safe operation.

Different types of non-functional requirements are shown in fig. 2.1.

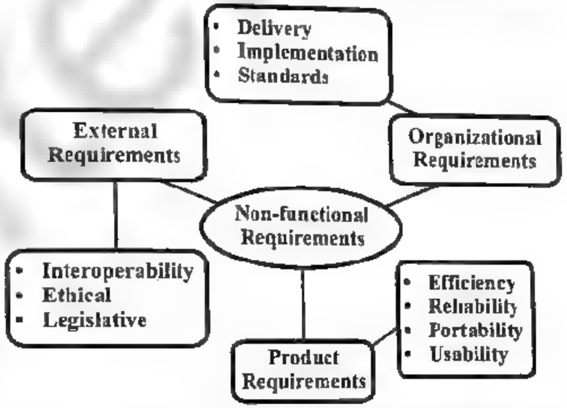


Fig. 2.1 Types of Non-functional Requirements

Q.4. How many types of requirements are possible and why ?
(R.G.P.V., Dec. 2015)

Ans. Software system requirements are often classified as functional, non-functional and domain requirements.

(i) Functional Requirements – Refer to Q.3.

(ii) Non-functional Requirements – Refer to Q.3.

(iii) Domain Requirements – Requirements which are derived from the application domain of a system instead from the needs of the users are called domain requirements. These requirements may be new functional requirements or specify a method to perform some particular computations. Moreover, these requirements include any constraint that may be present in the exiting functional requirements. It is important to understand these requirements because domain requirements reflect the fundamentals of the application domain. Also, if these requirements are not fulfilled, it may be difficult to make the system work as desired. A system can include a number of domain requirements.

Q.5. Explain requirement engineering with complete process. Also give its types.
(R.G.P.V., Dec. 2016)

Ans. Requirement engineering is a critical stage of the software process because errors at this stage inevitably lead to later problems in the system design and implementation. Fig. 2.2 illustrates the requirement engine process that leads to the production of a requirements document which is specification for the sytem. The requirement engineering process contains following phases –

(i) Feasibility Study – An estimate is made of whether the identi user needs may be satisfied using current software and hardware technolog The study will be decide if the proposed system will be cost effective fro business point of view and if it can be developed given existing budge constraints. A feasibility study should be relatively cheap and quick. The re should inform the decision of whether to go ahead with a more detailed analy

(ii) Requirements Elicitation and Analysis – This is the proces deriving the system requirements through observation of existing syste discussions with potential users and procurers, task analysis, etc. Here, o or more different system models and prototyes are developed. These help analyst understand the system to be specified.

(iii) Requirements Specification The information gathered du the analysis activity translated into a document that defines a set of requireme User requirements and system requirements are include in this document.

(iv) Requirements Validation – This activity checks the requireme for realism, consistency and completeness. Errors in the requirements docum are inevitably discovered during this process. It must then be modified correct these problems.

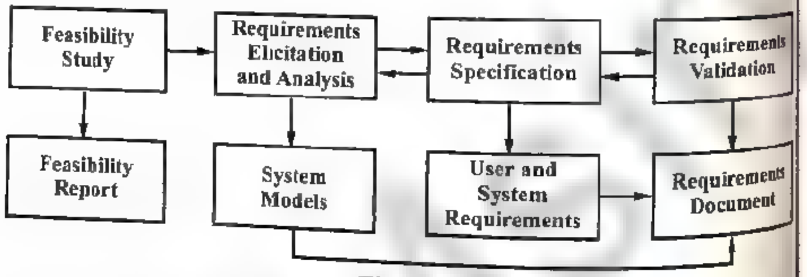


Fig. 2.2

Types – Refer to Q.4

Q.6. What is requirement analysis ? Explain the major activities requirement analysis.

(R.G.P.V., June 20

Ans. Requirement analysis is a software engineering process. It bridges gap between system level requirements engineering and software design (F 2.3). Requirements engineering activities provide the specification of softwa operational characteristics (function, data and behaviour), specify softwa interface with other system elements, and determine constraints that soft must satisfy. Requirements analysis permits the software engineer (someti

known as analyst in the role) to refine the software allocation and make models of the data, functional and behavioural domains that will be handled by software. Requirements analysis gives the software designer with a representation of information, function, and behaviour that can be translated to data, architectural, interface, and component-level designs. At last, the requirements specification gives the developer and the customer with the means to determine quality once software is built.

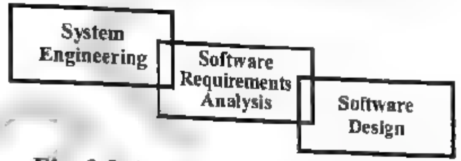


Fig. 2.3 Analysis as a Bridge between System Engineering and Software Design

Software requirements analysis may be split into five activities –

(i) Problem recognition (ii) Evaluation and synthesis
(iii) Modeling (iv) Specification
(v) Review.

In the beginning, the analyst studies the *System specification* (if any) and the *Software Project Plan*. It is significant to understand software in a system context and to review the software scope that was used to produce planning estimates. Next communication for analysis must be established so that problem recognition is ensured. The objective is to identify the basic problem elements as perceived by the customer/users.

The next major area of effort for analysis is the problem evaluation and solution synthesis. The analyst must define all externally observable data objects, evaluate the flow and content of information, define and elaborate all software functions, understand software behaviour in the context of events that affect the system, determine system interface characteristics, and uncover additional design constraints. Each of these tasks serves to describe the problem so that an overall approach or solution may be synthesized.

Upon evaluating current problems and desired information (input and output), the analyst starts to synthesize one or more solutions. To start, the data objects, processing functions, and behaviour of the system are defined in detail. Once this information has been obtained, basic architectures for implementation are considered. A client/server approach would seem to be suitable but does the software to help this architecture fall within the scope outlined in the *Software Plan* ? A database management system would seem to be needed, but in the user/customer's need for associativity justified ? The process of evaluation and synthesis continues until both analyst and customer feel confident that software can be sufficiently specified for subsequent development steps.

The analyst's main focus is on "what", not "how" throughout evaluation and solution synthesis. What data does the system generate and consume, what functions must the system do, what behaviours does the system show, what interfaces are established and what constraints apply ?

During the evaluation and solution synthesis activity, the analyst genera models of the system in an effort to better understand data and control fl functional processing, operational behaviour, and information content. T model works as a foundation for software design and as the basis for t generation of specifications for the software.

The detailed specification may not be possible at this stage. The custom may be unsure of precisely what is needed. The developer may be unsure th a certain approach will properly accomplish function and performance. F these, and many other reasons, an alternative approach to requirements analy known as prototyping, may be performed.

Q.7. List out the activities involved in software requirement analy What is requirement validation ? ***(R.G.P.V., Dec. 20***

Ans. Refer to Q.6 and Q.5 (iv).

Q.8. What types of models do we create during requirements analysi

Or

Compare functional and behavioural models. ***(R.G.P.V., Dec. 20***

Ans. We build models of function and behaviour during requirement analysi

(i) Functional Models – Software transforms information, and order to accomplish this, it must perform at least three generic functions input, processing and output. The software engineer focuses on probl specific functions when functional models of an application are created. T functional model starts with a single context level model, over a series iterations, more and more functional detail is provided, until a thorou delineation of all system functionality is represented.

(ii) Behavioural Models – Most software responds to events fr the outside world. This stimulus/response characteristics forms the basis the behavioural model. A computer program always exists in some state (e waiting, computing, printing, polling) that is changed when some event ta place. A behavioural model creates a representation of the states of the softw and the events that cause a software to change state.

Q.9. What are the different types of requirements gathering activi that the analysis use to gather requirements from a customer ?

(R.G.P.V., June 2012, 20

Or

What are the dimensions of requirements gathering ?

(R.G.P.V., May 20

Ans. The requirement process is the sequence of activities that requir be performed in the requirements phase and that culminate in producing a hi quality document containing the software requirement specification (SRS).

The requirement phase has three basic activities – problem or requirement analysis, requirement specification, and requirement validation. The first aspect deals with understanding the problem, the goals, the constraints, etc. Problem analysis begins with some general statement of need or a high-level problem statement. During analysis the problem domain and the environment are modeled in an effort to understand the system behaviour, constraints on the system, its inputs and outputs, etc. The main objective of this activity is to get a thorough understanding of what the software needs to provide. The understanding obtained by problem analysis makes the basis of the second activity requirements specification – in which the focus is on specifying the requirements in a document. During this activity, issues like representation, specification languages, and tools, are addressed. As analysis provides large amounts of information and knowledge with possible redundancies. An important goal of this activity is to properly organize and describe the requirements. The final activity focuses on validating that what has been specified in the SRS is of good quality. The requirement process ends with the production of the validated SRS.

Q.10. What are the different sources of understanding software requirements ?

Ans. The requirements specification of the software provides a base for developing the system and this is one of the most crucial steps in SDLC. Although the stakeholder is the ultimate source of the requirements, you cannot depend on the specification stated by a single source. For single source requirements, there will be almost no possible validation of the specifications because no comparable check will be present for the stipulated specifications. Hence, professionals gather specifications from different sources, including the customer, consumer, problem domain experts, connected domain experts, prospective users, operators, experienced developers, and even the critics of the system. A set of knowledge data is prepared from the execution of an existing manual or semi-automated system. The feedback of the owner, users, operators, other workers, and beneficiaries is also gathered and their suggestions and expectations about the new system are recorded. The gathered data is evaluated and validated collectively and refined in consultation with the people involved.

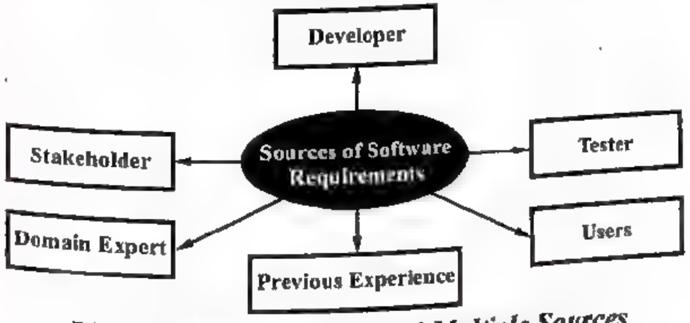


Fig. 2.4 Requirements Need Multiple Sources

In general, the sources of requirement are the following (see fig. 2.

(i) ***Stakeholders/Buyers*** – These are the individuals responsib accepting and owning the software. They may be individual pers organizations, trusts or even the government or public of a country.

(ii) ***Users/Beneficiaries*** – These are the end-users of the prod for whom the product is intended.

(iii) ***Operators*** – They are the persons who work on the softwa make the services of the software available to its beneficiaries or the end-us

(iv) ***Domain Experts*** – They are professionals with experienc expertise of the domain in which the software provides its services, insurance, financials, banking, communication, data transfer, networking Domain experts unwind the hidden or unseen probable requirements or involved in product development.

(v) ***Developer*** – The software engineers responsible for develo the software to make it provide the expected services. They are respon for software design, prototype development, and technical feasibility. T work closely with the end-users, buyers, and application experts.

(vi) ***Automated Tools*** – In the new-generation informa technology and software development paradigm, many automated and s automated tools are available that allow elicitation and management o requirement of a system to be built. Such software also provides inputs t requirements of the system/software.

(vii) ***Past Experience/Case Studies*** – An organization workin the similar or same domain may provide its past experience or even docum case studies. This helps have a clearer picture of the requirements, which otherwise be left hidden.

(viii) ***Connected People/Machine/Environment*** – The peopl environmental factors associated with the software and IT domain may give a information about the constraints and risks involved with the development, its im on the environment, and the implications of the environment over the softwa

(ix) ***Tester*** – Testers are good source of information about the pred behaviour of the user in a particular situation or the state of the system. Organiza cannot have constant interaction with actual users for their inputs. In such c testers may use their experiences and analytical skills to provide an input.

Q.11. What do you mean by requirements elicitation ?

Or

Why requirement elicitation is difficult ? ***(R.G.P.V., June***

Ans. Requirements elicitation, also called as ***requirements captu requirements acquisition***, is a process of collecting information about soft requirements from different individuals, such as users and other stakehol Stakeholders are individuals who are affected by the system, direct

indirectly. They include project managers, marketing personnel, consultants, software engineers, maintenance engineers, and the user.

Various issues may arise during requirements elicitation and may cause difficulties in understanding the software requirements. Some of the problems are –

(i) ***Problem of Scope*** – This problem arises when the boundary of software (i.e., scope) is not defined properly. Due to this, it becomes difficult to identify objectives as well as functions and features to be accomplished in the software.

(ii) ***Problem of Understanding*** – This problem arises when users are not certain about their requirements and thus are unable to express what they require in a software and which requirements are feasible.

(iii) ***Problem of Volatility*** – This problem arises when requirements change over time.

Q.12. Discuss the elicitation techniques of software requirements.

Or

Explain the ways and means for collecting the software requirements. ***(R.G.P.V., May 2019)***

Ans. This technique helps the stakeholders to clearly express their requirements by stating all the important information. The commonly followed elicitation techniques are –

(i) ***Interviews*** – It is a conventional way of eliciting requirements, which helps software engineers, users, and the software development team to understand the problem and suggests solutions for it. For this, the software engineer interviews the users with a series of questions. When an interview is conducted, rules are established for users and other stakeholders. In addition, an agenda is prepared before conducting interviews, which includes the important points to be discussed among users and other stakeholders.

An effective interview should have the following characteristics –

(a) Individuals involved in interviews should be able to accept new ideas. Also, they should focus on listening to the views of stakeholders related to requirements and avoid biased views.

(b) Interviews should be conducted in a defined context to requirements rather than in general terms. For this, users should start with a question or a requirements proposal.

(ii) ***Scenarios*** – These are descriptions of a sequence of events, which help to determine possible future outcomes before a software is developed or implemented. Scenarios are used to test whether the software will accomplish user requirements or not. Also, scenarios help to provide a framework for questions to software engineer about users tasks. These questions are asked from users about future conditions (what-if) and procedures (how) in which they think tasks can be completed.

Generally, a scenario includes the following information –

(a) Description of what users expect when the scenario starts.

(b) Description of how to handle the situation when a softw is not functioning properly.

(c) Description of the state of the software when the scenario e

(iii) Prototypes – Prototypes help to clarify unclear requireme Like scenarios, prototypes also help users to understand the information need to provide to the software development team.

(iv) Quality Function Deployment (QFD) – This deploym translates user requirements into technical requirements for the software this. QFD facilitates the development team to understand what is valuabl users so that quality can be maintained throughout software developmen

QFD identifies some of the common user requirements, which are as follo

(a) General Requirements – These requirements desc the system objectives, which are determined by various requirements elicita techniques. For example, graphical displays requested by users, spe software functions, etc.

(b) Expected Requirements – These requirements are imp to software, as users consider them to be fundamental requirements, which be accomplished in the software and hence do not express them. For exam ease of software installation, ease of software and user interaction, etc.

(c) Unexpected Requirements – These requirements spe the requirements that are beyond user expectations. These requirement not requested by the user but if added to the software, they become an additi feature of the software. For example, word processing software with additi capabilities, such as page layout capabilities along with the earlier feature

Q.13. List out requirements elicitation techniques. Which one is m popular and why ? ***(R.G.P.V., Dec. 2***

Ans. Refer to Q.12.

Interviews are probably one of the most popular elicitation techni Having one-to-one or one-to-many discussions with clients or users is a pop start for the requirement elicitation stage on many projects.

ANALYSIS MODELING FOR FUNCTION-ORIENTED AN OBJECT-ORIENTED SOFTWARE DEVELOPMENT, USE CA MODELING

Q.14. Define the term analysis and modeling. State the objecti analysis and modeling. ***(R.G.P.V., June 2***

Ans. **Analysis** – Requirement analysis is defined as the process of stu user needs to arrive at a definition of a system, hardware, or soft requirements. Requirement analysis helps to understand, interpret, cla and organize the software requirements in order to assess the feasib

completeness and consistency of the requirements. Following are the several other tasks that are performed using requirement analysis –

(i) To understand the problem for which a software is to be developed.

(ii) To develop an analysis model to analyze the requirements in the software.

(iii) To detect and resolve conflicts that arise due to unclear and unspecified requirements.

(iv) To determine operational characteristics of a software and how they interact with the environment.

Modeling – Modeling is a concept of abstraction in analysing or designing a system. This abstraction is a means of revealing the information from a particular point of view and hiding unnecessary information from this viewpoint. Modeling may be represent in terms of a set of diagrams, phrases or in some other forms, through which it presents the relevant information. A model is a representational abstraction containing a set of logical and quantitative relationship between the members of a set of variables or primitives. A model aims to provide an augmentative framework for applying logic and mathematics that can be independently evaluated and that can be applied for reasoning in the range of situations.

Objectives of Analysis and Modeling – Objectives of analysis and modeling are given below –

(i) To obtain a clear understanding of the needs of the clients and the users.

(ii) To establish a basis for the creation of a software design.

(iii) To devise a set of valid requirement after which the software can be built.

Q.15. Explain data flow diagram (DFD). ***(R.G.P.V., June 2016)***

Ans. ***Data flow diagram***, also called ***data flow graph***, is commonly used during problem analysis. They are quite general and are not limited to problem analysis. A data flow diagram (DFD) is a graphical representation of information-flow and the transforms that are applied as data move from input to output. It was in use long before the software engineering discipline began.

The DFD may be used to represent a system or software at any level of abstraction. It shows the flow of data through a system. It views a system as a function that transforms the input into desired outputs, which any complex system cannot perform in a single step. The DFD aims to capture the transformations that take place to the input data to eventually get the output.

DFDs may be divided into different levels that represent increasing information flow and functional detail. A level 0 DFD is called a ***fundamental system model*** or a ***context model***. It represents the whole software element as a bubble with incoming and outgoing arrows representing input and output o respectively. More processes i.e., bubbles and information flow paths ca represented by further partitioning the level 0 DFD. For example, a level 1 D might have six or seven bubbles with interconnecting arrows, each of w represents a subfunction of the overall system. Fig. 2.5 shows this conce

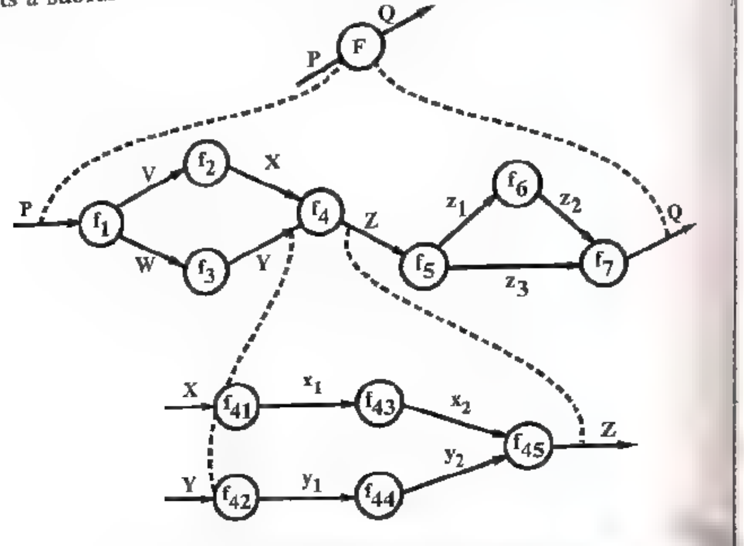


Fig. 2.5 Information Flow Refinement

Fig. 2.5 shows a fundamental model for system F which indicates P primary input and Q as an ultimate output. f_1 to f_7 are the transforms ge after the refinement of the F model. It should be noted that information f continuity must be maintained i.e., each refinement must have the same i and output. Further refinement of f_4 shows details as the transforms f_4 f_{45}. At both refinements, input is (X, Y) and output is (Z).

Q.16. What are the main shortcomings of data flow diagram (DFD a tool for performing structured analysis ? ***(R.G.P.V., June 20***

Ans. The main shortcomings of data flow diagram are as follows–

(i) *DFD Leave Ample Scope to be Imprecise.* In the DFD mo we judge the function performed by a bubble from its label. Although, a s label may not capture the whole functionality of a bubble.

(ii) *Control Aspects are not Defined by a DFD.* The sequence which inputs are taken and outputs are generated by a bubble is not specifie DFD model does not specify the order in which the different bubbles are exec

(iii) The data flow diagrams are highly subjective.

(iv) The data flow diagrams do not provide any guideline a how the process should be decomposed to get next level data flow diag

(v) The data flow diagram do not specify the dependencies among the task. Under such cases, DFD do not provide any useful information.

(vi) Data flow diagram undergoes lot of alteration before going to users, so makes the process little slow.

(vii) Physical considerations are left out.

(viii) The difference in symbols in different DFD models can create confusion.

Q.17. Define DFD (data flow diagram). Also describe the limitations of DFD for performing structural analysis. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.15 and Q.16.

Q.18. Differentiate between the DFD and flowchart. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. A DFD is not a flowchart. A DFD represents the flow of data, while a flowchart shows the flow of control. A DFD does not represent procedural information. So, while drawing a DFD, one must not get involved in procedural details, and procedural thinking must be consciously avoided. For instance, considerations of loops and decisions must be ignored. In drawing the DFD, the designer has to specify the major transforms in the path of the data flowing from the input to output. How those transforms are performed is not an issue while drawing the data flow graph.

Q.19. Consider an 'order processing system' that receives the orders through its salesman and sends good via transport and maintains track record of every order, including order receiving dates, delivery dates, invoice details and so on.

Use your imaginations to add more functionality to this system and develop context level 1 and level 2 data flow diagrams to model this 'order processing system'. ***(R.G.P.V., June 2006)***

Ans. A system development of invoice refers to an organization creating invoice manually. Now organization needs automation of invoice creation system. Therefore, invoice creation is the key activity of system or name of the information system. Now we prepare a DFD of level 0 i.e., context diagram.

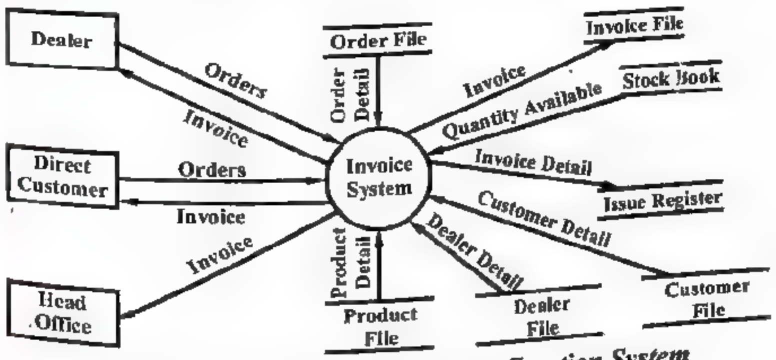


Fig. 2.6 Level 0 DFD of Invoice Creation System

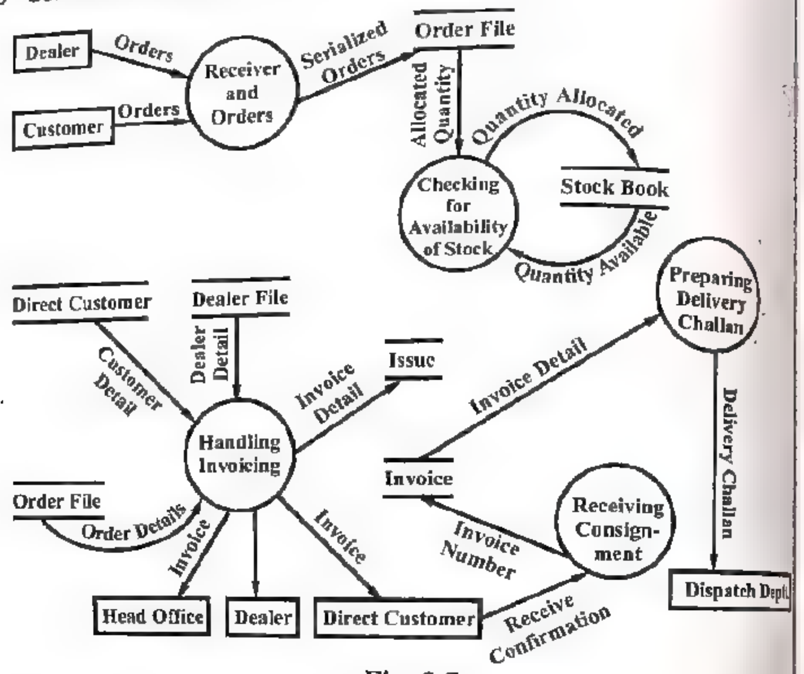


Fig. 2.7

Preparation of invoice can have several activities which can be hand as an individual process means one main process can have five subproces (see fig. 2.7).

(i) Receive and serialize orders

(ii) Checking for availability of stock for orders

(iii) Handling invoicing

(iv) Preparing the delivery challan

(v) Receiving consignment received confirmation.

By listing out the above five functions, we have named the list of processes or the bubbles that could form the lower level i.e., below the hig level process which we would call as "preparation of invoices".

Q.20. Explain the elements of analysis modeling. Draw a DFD admission system of your university. ***(R.G.P.V., Dec. 20***

Or

Explain analysis modeling. What are its elements ? Also expl functional modeling. ***(R.G.P.V., Dec. 2006, June 20***

Ans. **Analysis Modeling** – The written word is a nice vehicle communication, but it is not essentially the best way to represent requirements for computer software. Analysis modeling uses a combina of text and diagramatic forms to show requirements for data, function, behaviours in a way that is relatively simple to understand, and more impor straightforward to review for completeness, correctness, and consistenc

A software engineer (also called an analyst) construct the model using requirements obtained from the customer. Several diagramatic formats are used to model data, functional, and behavioural requirements. Data modeling defines data objects, attributes, and relationships. Functional modeling specifies how data are transformed within a system. Behavioural modeling shows the impact of events. Once preliminary models are produced, they are refined and analyzed to determine their clarity, completeness, and consistency. A specification incorporating model is produced and then validated by both software engineers and customers.

In fig. 2.8, the analysis model serves as the connection between the system description and design model. System description gives information about the whole functionality of the system, which is obtained by implementing the software, hardware, and data. Moreover, the analysis model provides the software design, in the form of a design model, which provides information about the software's architecture, user interface and component level architecture.

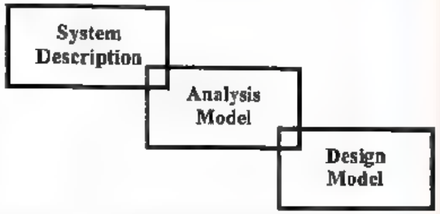


Fig. 2.8 Analysis Model as Connector

Elements of the Analysis Model – The analysis model must meet three primary objectives

(i) To describe the customer requirements.

(ii) To provide a basis of creating software design.

(iii) To specify a set of requirements that can be validated after software has been built.

The analysis model derived during structured analysis takes the form shown in fig. 2.9 to accomplish these objectives.

The analysis model consists of the following elements as shown in fig. 2.9 –

(i) Data dictionary	(ii) Entity relationship diagram
(iii) Data-flow diagram	(iv) State-transition diagram
(v) Data object description	(vi) Process specification
(vii) Control specification.	

(i) Data Dictionary – It lies at the core of analysis model. It is the repository of data objects consumed or produced by the software.

The core is surrounded by three diagrams –

(ii) Entity Relationship Diagram – It shows the relationships between data objects. It is the notation used for conducting data modeling. The attributes of each data object given in the ERD can easily be described using a data object description.

(iii) Data-flow Diagram – It fulfills two tasks –

(a) It provides an indication of data transformation as they m through the system.

(b) It shows the functions of data transformation.

The DFD also provides the information used during the information do analysis. It serves as a basis for function modeling.

(iv) State-transition Diagram – It indicates the system behavi as a consequence of external events. For that, it represents the various mod of behaviour known as ***states*** of the system. Also, STD represents the man in which the transitions from one state to another are made. The STD provi the foundation to behavioural modeling.

(v) Data Object Description – The attributes of each data obj given in ERD can be described using a data object description.

(vi) Process Specification – The description of the DFD function is contained in a process specification.

(vii) Control Specification – Extra information about the control aspects of the software is contained in the control specification.

Analysis model encompasses each of the diagrams, specifications, descriptions, and the dictionary in fig. 2.9.

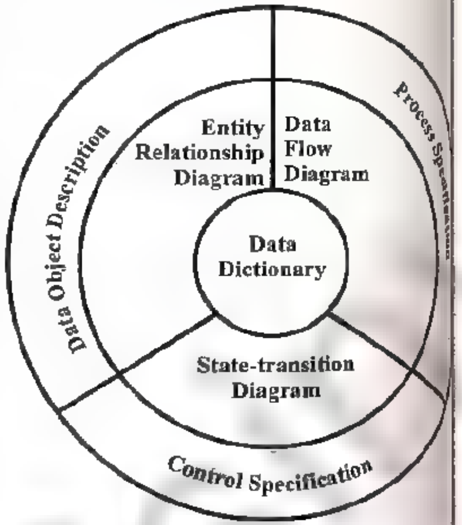


Fig. 2.9 The Structure of Analysis Mo

DFD for Admission System of a University – Fig. 2.10 shows the level 0 DFD for admission system.

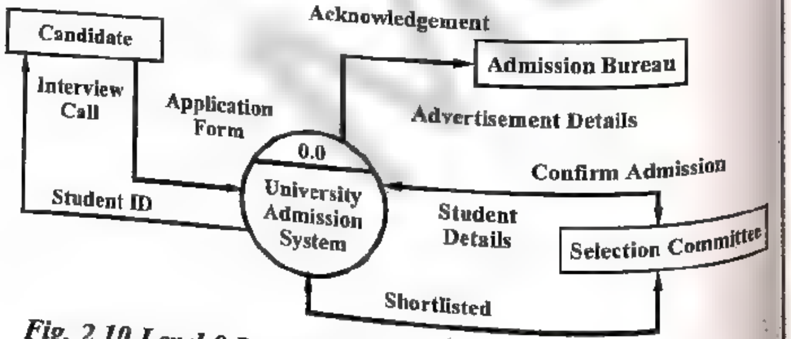


Fig. 2.10 Level 0 Data Flow Diagram of Admission System

The detailed DFD for admission system is shown in fig. 2.11. University admission system can be defined as a system that keeps tracks of the entire admission procedure for a candidate seeking admission to a particular college in a specific stream. The system is primarily concerned with the issue of

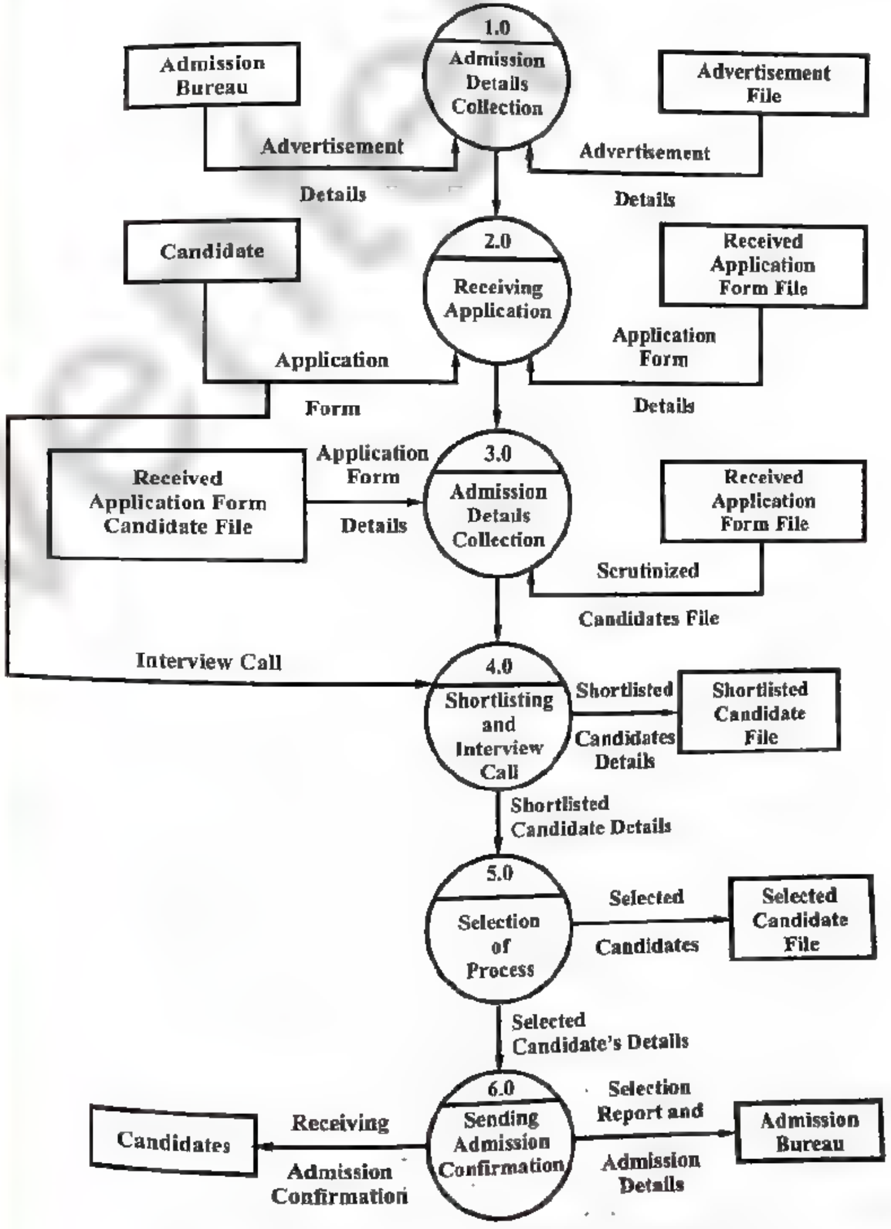

Admission Bureau
1.0
Admission Details Collection
Advertisement File
Advertisement Details
Advertisement Details
Candidate
2.0
Receiving Application
Received Application Form File
Application Form
Application Form Details
Received Application Form Candidate File
Application Form Details
3.0
Admission Details Collection
Received Application Form File
Scrutinized Candidates File
Interview Call
4.0
Shortlisting and Interview Call
Shortlisted Candidates Details
Shortlisted Candidate File
Shortlisted Candidate Details
5.0
Selection of Process
Selected Candidates
Selected Candidate File
Selected Candidate's Details
6.0
Sending Admission Confirmation
Candidates
Receiving Admission Confirmation
Selection Report and Admission Details
Admission Bureau

Fig. 2.11 Detailed DFD for Admission System

advertisement for admission, receiving applications from the inter candidates, shortlisting of the deserving candidates, students are interv by the selection committee. After which according to the student's perform in the interview, the students admission are finally confirmed and the sam apprised to the candidate with is ID.

This system thus helps in the conjunction of the two entities, nam CANDIDATE (Student) and COLLEGE.

Functional Modeling – The functional model describes the computa that take place within a system. It looks at the functionality of the system other words, the functional model of a system specifies what happens in system.

A functional model of a system specifies how the output values are comp in the system from the input values, without considering the control aspect the computation. This represents the functional view of the system. T functional model of a system can be represented by a data flow diagram (DF

Also refer to Q.21.

Q.21. Explain functional modeling and information flow modeli

Ans. In functional models, software transforms information. And in or to accomplish this, it must perform at least three generic functions – in processing, and output. When functional models of an application are crea the software engineer emphasizes on problem specific functions. The functi model starts with a single context level model (i.e., the name of the soft to be built). Over a series of iterations, more and more functional detai given, until a thorough delineation of all system functionality is represent

The transformation of information is done because it flows throug computer-based system. The system takes input in different forms; app hardware, software, and human elements to transform it; and produces ou in different forms. The transform(s) or function may composed of a si logical comparison, a complex numerical method, or a rule-inference appr of an expert system. Output may light a LED or provide a 200 page rep Instead, we can make a model or ***flow model*** for any computer-based sys regardless of size and complexity.

Structural analysis started as an information flow modeling techniq computer-based system can be modeled as an information transform fu as shown in fig. 2.12.

A rectangle represents an external entity. That is, a system element, hardware, a person or another system that provides information for transform by the software or receives information provided by the software. A cir used to represent a process or transform or a function that is applied to data changes it in some way. An arrow is used to represent one or more data it

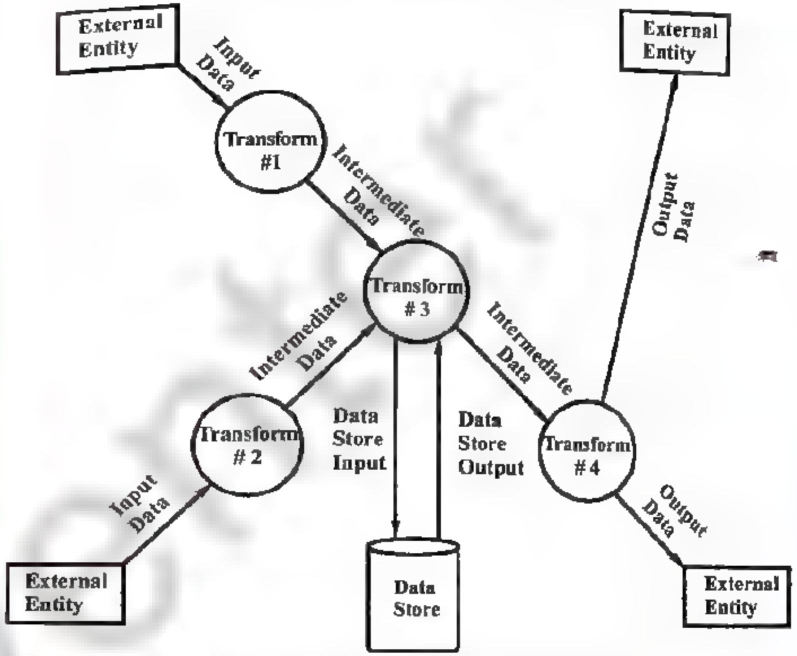


Fig. 2.12 Information-flow Model

All arrows should be labeled in a DFD. The double line is used to represent data store. There may be implicit procedure or sequence in the diagram but explicit logical details are generally delayed until software design.

Q.22. What are the elements of analysis modeling, explain each of them in brief ? ***(R.G.P.V., June 2015)***

Ans. Refer to Q.20.

Q.23. Explain Ward and Mellor extensions for real-time systems.

Ans. To accommodate demands imposed by the analysis of real-time software, Ward and Mellor extend basic structured analysis notation as follows –

(i) Information flow is collected or generated on a time-continuous basis.

(ii) Control information is passed throughout the system and associated control processing.

(iii) Several multiple instances of the same transformation are sometimes found in multitasking environment.

(iv) System have states, among which a mechanism causes transition.

In most of real-time applications, the system monitors time-continuous information produced by some real-world process. Conventional data flow notation does not provide any distinction between discrete data and time-continuous data. Fig. 2.13 shows one extension to basic structured analysis notation, which provides a mechanism for representing time-continuous data flow.

As we can see in fig. 2.13, the double headed arrow represents time-continuous flow, while single headed shows discrete data flow. In this figure, temperature set point is a single value for temperature, previously set, while monitored temperature is measured continuously. Corrected value is a time-continuous output, produced at last.

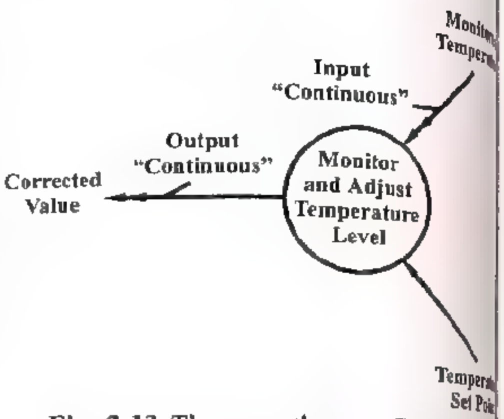


Fig. 2.13 Time-continuous Data Flow

Fig. 2.14 depicts control flow and processing, using Ward and Mel notation.

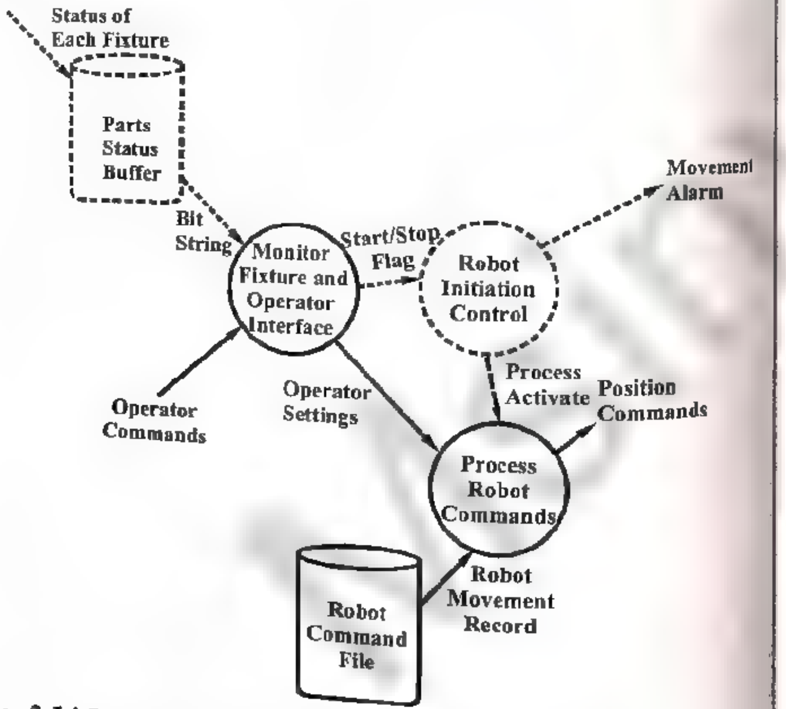


Fig. 2.14 Data and Control Flows using Ward and Mellor Notation

A top-level view of a data and control flow for a manufacturing cel shown in fig. 2.14. In the fig. 2.14, a status bit is set within a "parts sta buffer", which is a control store, since the components to be assembled b

robot are placed on fixtures. This status bit indicates that whether a particular component is present or not. Event information, present within the parts status buffer, is passed as a bit string to a process called "monitor fixture and operator interface", which reads operator commands only when the control information i.e., bit string, indicates the presence of components in all fixtures. An event flag called "start/stop flag" is sent to robot initiation control, which is a control process that enables further command processing. The "process activate" event is sent to "process robot commands", and as a consequence of which other data flows take place.

In some cases, multiple instances of the same control or data transformation process may take place in a real-time system. This can happen in a multitasking environment. For example, several "part status buffer" may be monitored so as to signal different robots at the suitable time. Also, each robot may have its own robot control system. The Ward and Mellor notation, representing multiple equivalent instances, simply overlays process bubbles to show multiplicity.

Q.24. Explain Hatley and Pirbhai extensions for real-time systems.

Or

What is the difference between data flow and control flow models with examples ? ***(R.G.P.V., Dec. 2003)***

Ans. The Hatley and Pirbhai extension to structured analysis notation concentrates less on the generation of additional graphical symbols and more on representing and specifying the control-oriented software aspects. The dashed arrow is used to show control or event flow. Hatley and Pirbhai suggest to represent dashed and solid notation separately, thus, defined a control flow diagram. The control flow diagram (CFD) contains the same processes as the data flow diagram (DFD), but the difference is that it represents the control flow rather than data flow. In place of indicating control processes directly within a flow model, a notational reference such as a solid bar, to a control specification is used.

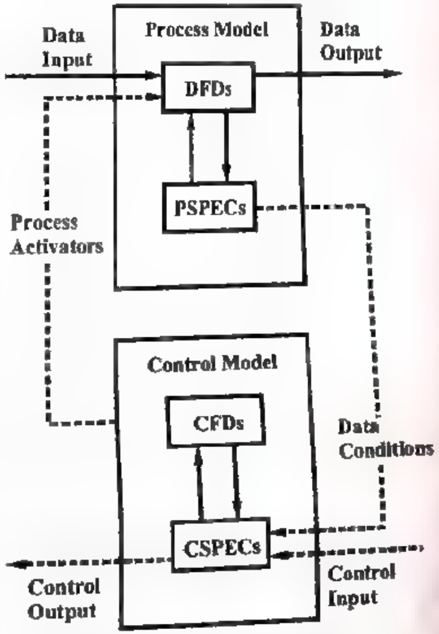
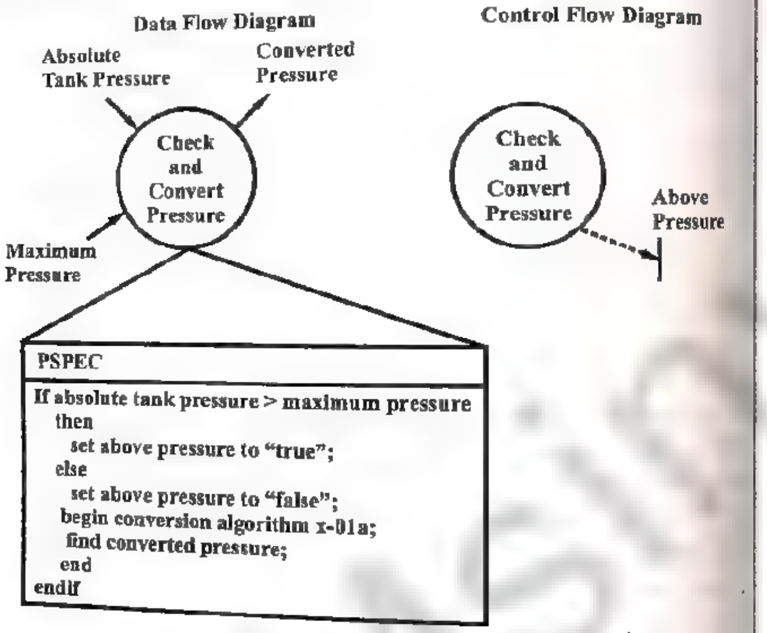


Fig. 2.15 The Relationship between Data and Control Models

Hatley and Pirbhai develop a model of a real-time system. Here, data fl diagrams are used to show data and the processes for manipulating it, wh control flow diagrams represent the events flow among processes and sh those external events that cause several processes to be activated. Fig. 2.1 shows the interrelationship between the process and control models.

Observe the fig. 2.15, the control model is connected to the proce model through process activators having activation information from t CSPEC, and the process model is connected to the control model throu data conditions.

Fig. 2.16 shows the situation of occurring data condition whenever d input to a process result in control output.



```
If absolute tank pressure > maximum pressure
  then
    set above pressure to "true";
  else
    set above pressure to "false";
  begin conversion algorithm x-01a;
    find converted pressure;
  end
endif
```

Fig. 2.16 Data Conditions

Part of a flow model is for an automated monitoring and control sys for pressure vessels in an oil refinery. The PSPEC, shown in fig. 2.16, desc the algorithm which is implemented by the process "check and con pressure". According to it, when the absolute tank pressure is greater tha allowable maximum pressure, an event, called "above pressure" is gener It should be noted that, using Hatley and Pirbhai notation, the data flow shown as part of a DFD, in contrast the control flow is separate being a of a control flow diagram. Observe the vertical solid bar into which "above pressure" event flow is a pointer to the CSPEC. Thus, to see consequence of this event, CSPEC must be checked.

Q.25. What is object-oriented modeling ? What are the steps involved in object-oriented analysis ?

Or

Explain the steps involved in object-oriented modeling.

(R.G.P.V., June 2017)

Ans. The object-oriented modeling defines a system as a set of objects, which interact with each other by the services they provide. In addition, objects interact with users through their services so that they can avail the required services in the system.

Object-oriented analysis consists of a number of steps, which are as follows –

(i) Identifying Objects – While performing an analysis, an object encapsulates the attributes on which it provides services. Note that an object represents entities in a problem domain. The identification of the objects starts by viewing the problem space and its description. Then, a summary of the problem space is gathered to consider the 'nouns'. Nouns indicate the entities used in problem space and which will further be modelled as objects. Some examples of nouns that can be modelled as objects are structures, events, roles and locations.

(ii) Identifying Structures – Structures depict the hierarchies that exist between the objects. Object modelling applies the concept of generalization and specialization to define hierarchies and to represent the relationships between the objects. A superclass is a collection of classes which can further be refined into one or more subclasses. A subclass can have its own attributes and services apart from the attributes and services inherited from its superclass. To understand generalization and specialization, consider an example of class 'car'. Here, 'car' is a superclass, which has attributes such as wheels, doors and windows. There may be one or more subclasses of a superclass. For instance, a superclass 'car' has subclasses 'Mercedes' and 'Toyota', which have the inherited attributes along with their own attributes, such as comfort, locking system, and so on.

It is essential to consider the objects that can be identified as generalization so that the classification of structure can be identified. In addition, the objects in the problem domain should be determined to check whether they can be classified into specializations or not. Note that the specialization should be meaningful for the problem domain.

(iii) Identifying Attributes – Attributes add details about an object and store the data for the object. For example, the class 'book' has attributes such as author name, ISBN, and publication house. The data about these attributes is stored in the form of values and are hidden from outside the objects. However, these attributes are accessed and manipulated by the service functions used f that object. The attributes to be considered about an object depend on the probl and the requirement for that attribute. For example, while modelling the stud admission system, attributes such as age and qualification are required for t object 'student'. On the other hand, while modelling for hospital managem system, the attribute 'qualification' is unnecessary and requires other attribu of class 'student', such as gender, height, and weight. In assence, it can be sa that while using an object, only the attributes that are relevant and required the problem domain should be considered.

(iv) Identifying Associations – Associations describe the relationsh between the instances of several classes. For example, an instance of cla 'university' is related to an instance of class 'person' by 'educates' relationshi Note that there is no relationship between the class 'university' and cla 'person'. However only the instance (s) of class 'person' (that is, student) related to class 'university'. This is similar to entity relationship modellin where one instance can be related by 1:1, 1:M, and M:M relationships.

An association may have its own attributes, which may or may not be prese in other objects. Depending on the requirement, the attributes of the associati can be 'forced' to belong to one or more objects without losing the informatio However, this should not be done unless the attribute itself belongs to that obje

(v) Defining Services – An object performs some services. The services are carried out when an object receives a message for it. Services a medium to change the state of an object or carry out a process. The services describe the tasks and processes provided by a system. It is importa to consider the 'occur' services in order to create, destroy, and maintain t instances of an object. To identify the services, the system states are defin and then the external events and the required responses are described. F this, the services provided by objects should be considered.

Q.26. Differentiate between function-oriented and object-orient modeling. ***(R.G.P.V., June 201***

Or

What is the difference between function-oriented and object-orien modelling ? Explain in detail. ***(R.G.P.V., June 201***

Or

Compare function-oriented and object-oriented software developme ***(R.G.P.V., Dec. 201***

Or

Differentiate between function-oriented and object-oriented softw development. ***(R.G.P.V., May 2018, 201***

Ans. Refer to Q.21 and Q.25.

Q.27. Why we develop the use case diagram ?

Ans. In the use case diagram, the utility of the use cases represented by the ellipses is clear. They along with the accompanying text description work as a kind of requirements specification of the system and the model depending on which all other models are produced. It means that, the use case model forms the core model to which all other models must follow. But, what about the actors ? How are they useful to system development ? The different types of users (actors) are used in implementing a security mechanism through a login system, so that each actor can invoke only those functionalities to which he is entitled to. Another important use is to design the user interface in the implementation of the use case targetted for each specific class of users who would use the use case. Another possible use is to prepare the documentation (for example, users, manual) targeted at each class of user. In addition, actors are useful in identifying the use cases and understanding the exact functioning of the system.

Q.28. What are the benefits of use case-driven development ?

Ans. Use case specifies the scenario in terms of transactions, users and system performance. It depends on user tasks and exp.ains user activities in the natural english language. It is helpful in

(i) Defining the scope of the use case, use cases and applications and system.

(ii) Measuring the size of the software development project in a number of use cases.

(iii) Tracking and monitoring progress of the development in terms of use cases.

(iv) Validating the requirement of the system as specified by the user.

(v) Designing test cases by each use case to guarantee completeness.

(vi) Ensuring complete system documentation, as all use cases are included with model, design, and details.

(vii) Reducing the user's requirement for training and hand-holding in learning to operate the system.

(viii) Using systems more efficiently.

(ix) Reducing time for system maintenance.

(x) Obtaining users fast acceptance of the system.

Q.29. Explain about use case modeling. ***(R.G.P.V., Dec. 2014)***

Or

Explain use case approach for specifying functional specifications. ***(R.G.P.V., June 2010, 2011)***

Or

Explain use case model. ***(R.G.P.V., June 2016)***

Ans. The use case model for a system is composed of a set of use cases.

All the use cases of a system can be found by asking the question "What all can the users do through the system?" Therefore, the use cases for the library information system (LIS) could be issue-book, query-book, return-book, create-member, add-book, etc.

In other words, the use cases correspond to the high-level functional requirements. It can also be said that the use cases divide the system behavior into transactions in such a way that each transaction does some useful action from the user's point of view. Each transaction may have multiple message exchanges between the user and the system to be complete.

The objective of a use case is to define a piece of coherent behavior hiding the internal structure of the system. The use cases do not reveal any specific algorithm to be used nor the internal data representation, internal structure of the software. In a use case, there is a sequence of interactions between the user and the system. Also for the same use case, there can be many distinct sequences of interactions. A use case is composed of one main line sequence and several alternate sequences. The main line sequence indicates the interactions between a user and the system that normally occur. The most frequently occurring sequence of interaction is the mainline sequence. For example, the use case supported by a bank ATM in the mainline sequence of the withdraw cash would be – the user inserts the ATM card, enters password, chooses the amount withdraw option, enters the amount to be withdrawn, completes the transaction, and collects the amount. There may be several variations to the main line sequence called ***alternate sequences***. Normally, a variation from the mainline sequence takes place when some specific conditions hold. Several variations or alternate sequences may occur for the bank ATM example. Consider, for example, the case where the password is invalid or the amount to be withdrawn is greater than the account balance. The mainline sequence and each of the alternate sequences corresponding to the invocation of a use case is known as a ***scenario*** of the use case.

A use case model is documented with the help of drawing a use case diagram and writing an accompanying text elaborating the drawing. In the use case diagram, an ellipse represents an use case having the name of the use case written inside the ellipse. A rectangle is used to enclose all the ellipses (i.e., use cases) of a system enclosed within a rectangle which represents the system boundary. The name of the system being modeled (for example, Library Information System) is written inside the rectangle.

A stick person icon is used to represent a user of the system. A stick person icon refers to an actor. An actor refers to a role played by a user with respect to the system use. A user may play the role of multiple actors. An actor can take part in one or more use cases. An actor and the use case can be connected by the line, which is known as the communication relationship. The relationship specifies that an actor makes use of the functionality provided by the use case.

Both human users and external systems are represented using stick person icons. When an external system is represented by a stick person icon, it is annotated by the stereotype <<external system>>.

The ***system boundary box***, specifies the scope of the system. Anything inside the box represents functionality that is in scope and anything outside the box is not. However, it is optional to draw the system boundary.

For example, the use case model for the Tic-tac-toe game software is depicted in fig. 2.17. In this software the use case is "play move". It is noted that the use case is not named "get-user-move", as "get-user-move" would not be suitable because this would represent the developer's perspective of the use case. From the users' perspective, the use cases should be named.

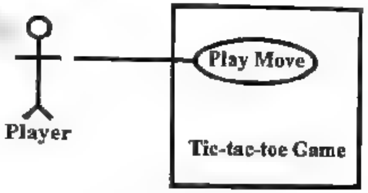


Fig. 2.17 Use Case Model

Q.30. Explain the use case diagram of ATM machine with neat diagram.
(R.G.P.V., May 2019)

Ans. An automated teller machine (ATM) or the automatic banking machine (ABM) is a banking subsystem (subject) that provides bank customers with access to financial transactions in a public space without the need for a cashier, clerk, or bank teller.

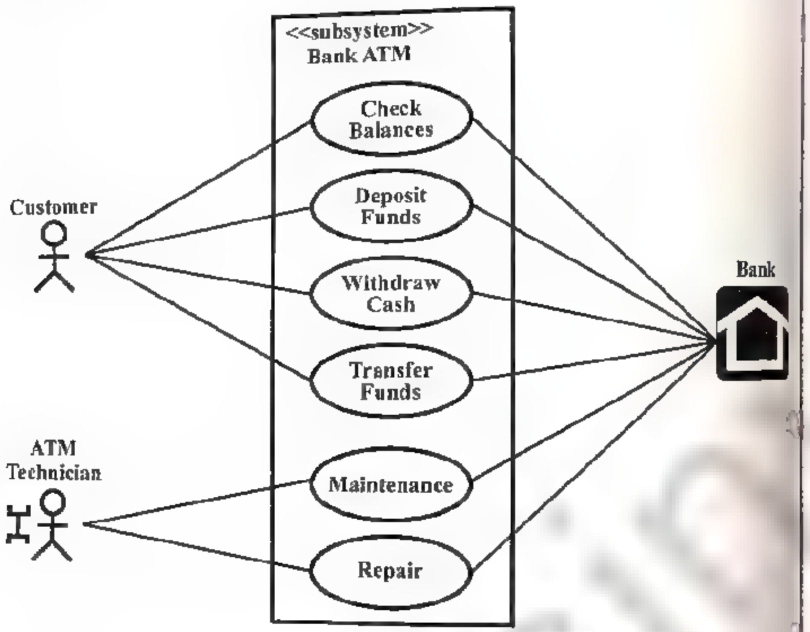
Customer (actor) uses bank ATM to check balances of his/her bank accounts, deposit funds, withdraw cash and/or transfer funds (use cases). ATM technician provides maintenance and repairs. All these use cases also involve bank actor whether it is related to customer transactions or to the ATM servicing.

On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). Customer authentication use case is required for every ATM transaction so we show it as include relationship. Including this use case as well as transaction generalizations make the ATM transaction an abstract use case.

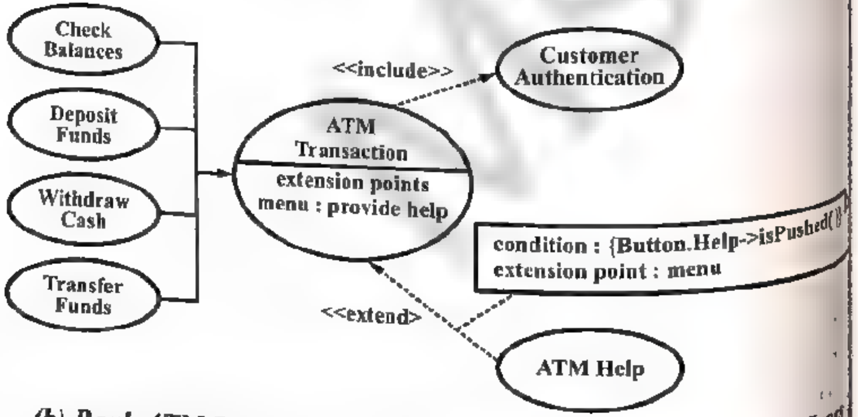
Customer may need some help from the ATM. ATM transaction use case is extended via extension point called menu by the ATM help use case whenever

ATM transaction is at the location specified by the menu and the bank customer requests help, e.g. by selecting help menu item.

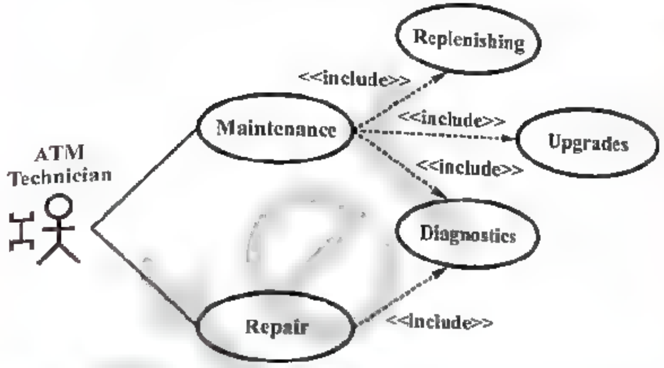
ATM technician maintains or repairs bank ATM. Maintenance use case includes replenishing ATM with cash, ink or printer paper, upgrades of hardware, firmware or software, and remote or on-site diagnostics. Diagnostics is also included in (shared with) repair use case. The example of use case diagram for bank ATM, transactions and maintenance as shown in fig. 2.18.



(a) Use Case Diagram for Bank ATM Subsystem



(b) Bank ATM Transactions and Customer Authentication Use Cases



(c) Bank ATM Maintenance, Repair, Diagnostics Use Cases
Fig. 2.18

SYSTEM AND SOFTWARE REQUIREMENT SPECIFICATIONS, REQUIREMENT VALIDATION, TRACEABILITY

Q.31. Define system specification.

Ans. The final work product produced by the system and requirements engineer is the system specification. The system specification serves as the basis for hardware engineering, software engineering, database engineering, and human engineering. It specifies the function and performance of a computer based system and the constraints that will govern its development. Each allocated system element is bounded by the specification. Also the system specification specifies the information that is input to and output from the system.

Q.32. Discuss an example of type of system where social and political factors might strongly influence the system requirements. Explain why these factors are important in your example ? ***(R.G.P.V., June 2017)***

Ans. Social and political factors strongly influence the system requirements. For example – Let person A be an employee of a company whose responsibility is to give the complete system requirement of the company. Now,

Social Factor – The friends of A forces A to give them exrtra or specific system requirements. This is what the poor pressure is and how social factor affects the system requirements.

Political Factor – The manager of company demanded for some specific system requirement which is of no need only to increase their influence in the organization.

This is how social and political factors strongly influence the system requirements.

Q.33. *Explain software requirement specification. What are th characteristics of good SRS ? Why SRS is known as black box specification* (R.G.P.V., Dec. 200

Ans. **Software Requirement Specification** – The output of th requirements phase of a software development process is the ***software requirement specification*** document. This is also referred to as requirement document. This document lays a foundation for software engineering activiti and is created when entire requirements are elicited and analyzed. SRS is formal document, which acts as representation of software that enables th users to review whether it (SRS) is according to their requirements or not. addition the requirements document includes user requirement for a system well as detailed specification of the system requirement.

IEEE defines software requirement specification as "a document th precisely and clearly describes each of the essential requirements of the softwa and the external interfaces. Each requirement is defined in such a way that achievement can be objectively verified by a prescribed method, for exampl inspection, demonstration, analysis, or test.

Essentially what passes from requirements analysis activity to th specification activity is the knowledge acquired about the system. The model is essentially a tool to help obtain a thorough and complete knowledge about proposed system. The SRS is written based on the knowledge acquired duri analysis. Since converting knowledge into a structured document is not straig forward specification itself is a major task, which is relatively independent.

The requirement document is used by various individuals in th organization. As shown in fig. 2.19 system customers needs, SRS to spec and verify whether requirements meet the desired needs or not. In additio SRS enables the managers to plan for the system development process System engineers require a requirements document to understand what syst is to be developed. They also require this document to develop validation tests for the required system. Requirements document is needed by system maintenance engineers to use the requirement and the relationship between its parts.

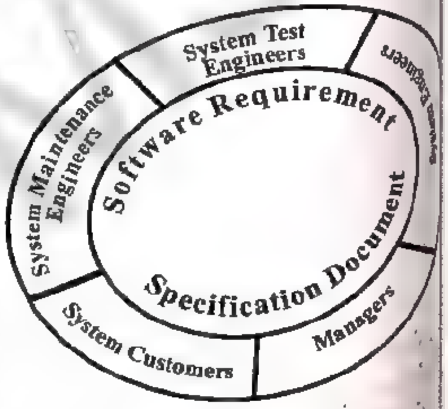

System Test Engineers
Engineers
Software Requirement Specification Document
System Maintenance Engineers
System Customers
Managers

Fig. 2.19 SRS Users

The requirement document has to define the requirements in precise detail for developers and testers. In addition it should also include information about possible changes in the system, which can help system designers avoid restricted decisions on design. SRS also helps maintenance engineers to adapt the system to new requirements.

Characteristics of a Software Requirement Specifications – The software requirement specifications should have certain properties to properly satisfy the basic goals and it should contain different types of requirements. Some desirable characteristics of the software requirement specifications are as follows –

(i) Correct – If every requirement included in the software requirement specification represents something required in the final system then an SRS is correct.

(ii) Complete – If everything the software is supposed to do and the responses of the software to all classes of input data are specified in software requirement specification then the SRS is complete. It ensures that every thing is indeed specified. As compare to correctness, completeness is most difficult property to establish.

(iii) Unambiguous – If and only if every requirement stated has one and only one interpretation, then the SRS is unambiguous. The requirements which are inherently ambiguous are often written in natural language. The using formal languages is the large effort required to write an SRS is the major disadvantage, and also the high cost of doing so, and increased difficulty reading and understanding formally stated requirements.

(iv) Verifiable – The software requirement specification is verifiable if and only if each state of requirement is to be verifiable. If there exists some cost-effective process which can check whether the final software meets the requirements then a requirement is verifiable. For verifiability the unambiguity is very essential.

(v) Consistent – If there is no requirement that conflicts with another then an SRS is consistent. To refer the same object, different requirements may use different terms. There may be logical or temporal conflict between requirements causing inconsistencies. It occurs when the SRS contains two or more requirements whose logical or temporal characteristics cannot be satisfied together by any software system.

(vi) Ranked for Importance and/or Stability – For importance and/ or stability an SRS is ranked, if for each requirement the importance and the stability of the requirement are indicated. The changes of it changes in future is reflected by stability of a requirement.

(vii) Modifiable – If the structure and style of SRS are in such way that, any change can be made easily while preserving completeness and consistency then the software requirement specification is modifiable. The resulting SRS will be inconsistent if only one occurrence of the requirement is modified.

(viii) Traceable – If the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development then an SRS is traceable. The traceability can be divided in two types, forward traceability and backward traceability. The forward traceability is that in which each requirement should be traceable to some design and code elements, while the backward traceability requires that it is possible to trace design and code elements to the requirements they support. The traceability supports the verification and validation.

SRS as Black Box Specification – SRS is called as black box specification because it addresses only about what the proposed system should do without emphasizing on how to acquire it. It is the output of the analysis activity and appears very early in the development process. Essentially, what passes from requirements analysis activity to the specification activity is the knowledge acquired about the system. The modelling is essentially a tool to help gain thorough and complete knowledge about the proposed system. The SRS is written on the basis of the knowledge obtained during analysis. Since converting knowledge into a structured document is not straightforward, specification itself is a major task, which is relatively independent.

A result of this is that it is relatively less important to model "completely" compared to specifying completely. As the main purpose of analysis is problem understanding, while the fundamental purpose of the requirements phase is to produce the SRS, the complete and detailed analysis structures are not crucial as the proponents of a technique make it out to be. In fact, it is possible to develop the SRS without using formal modeling technique. The basic purpose of the structures used in modeling is to support in knowledge representation and problem partitioning, the structures are not an end in themselves.

Q.34. Explain software requirement specification. ***(R.G.P.V., May 201***

Or

What is software requirement specification (SRS) ? State its principles and characteristics. ***(R.G.P.V., May 201***

Ans. Refer to Q.33.

Q.35. List out the importance of software requirement document in brief. ***(R.G.P.V., June 201***

Ans. Refer to Q.33.

Q.36. List five desirable characteristics of a good software requirement specification (SRS) document. ***(R.G.P.V., June 2013)***

Or

What are the characteristics of good SRS ? ***(R.G.P.V., Dec. 2014)***

Or

Explain the different characteristics of software requirement specification. ***(R.G.P.V., June 2015)***

Or

Explain the nature of SRS. ***(R.G.P.V., June 2017)***

Ans. Refer to Q.33.

Q.37. What do you mean by system specification, requirement specification, software specification ? Explain in brief.
(R.G.P.V., Dec. 2009, June 2013)

Ans. **System Specification** – Refer to Q.31.

Requirement Specification – Refer to Q.33.

Software Specification – Software specification is an activity that is intended to establish what services are required from the system and the constraints on the system's operation and development. This activity is often called ***requirements engineering***. Requirements engineering is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation.

The requirement engineering process is shown in fig. 2.20. This process leads to the production of a requirements document which is the specification for the system. Requirements are usually presented at two levels of detail in this document. End-users and customers need a high-level statement of the requirements. System developers need a more detailed system specification.

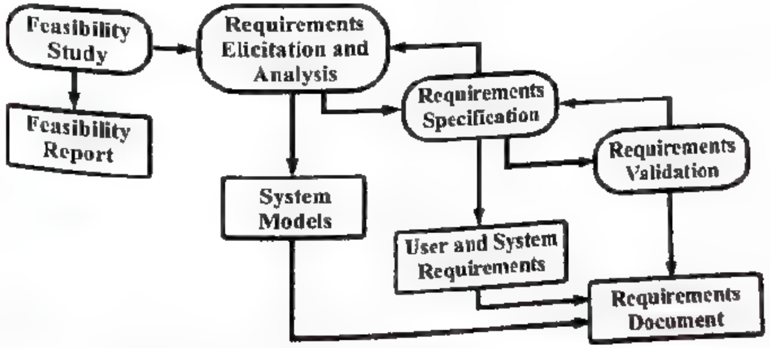


Fig. 2.20 The Requirements Engineering Process

There are four main phases in the requirements engineering process.

(i) ***Feasibility Study*** – An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study will decide if the proposed system will be cost-effective from business point of view and if it can be developed given existing budgetary constraints. A feasible study be relatively cheap and quick. The result should inform the decision of whether to go ahead with a more detailed analysis.

(ii) ***Requirements Elicitation and Analysis*** – This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, etc. This may involve the development of one or more different system models and prototypes. These help the analyst understand the system to be specified.

(iii) ***Requirements Specification*** – Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of system. System requirements are a more detailed description of the functionality to be provided.

(iv) ***Requirements Validation*** – This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

Q.38. What is SRS ? What is the need of SRS ? What are the advantages of SRS document ? ***(R.G.P.V., Dec. 201***

Ans. SRS – Refer to Q.33.

Need of SRS – Software systems mainly focus on the need of a client who either wants to automate an existing manual system or desires a new software system. The software system itself is created by the developer. At last, the completed system will be used by the end users. Therefore, in a new system there are mainly three major parties interested – the client, the developer and, the users. Somehow the requirements for the system that will satisfy the needs of the clients and the concerns of the users have to be communicated to the developer. But the difficulty is that the client usually does not understand software or the software development process and the developer often does not understand the client's problem and application area. This causes a large communication gap between the parties involved in the development. To bridge this communication gap is a basic purpose of software requirements specification. SRS forms the basis of software development. SRS is a medium through which the client and user needs are accurately specified. A good SRS should satisfy all the parties.

Another important purpose of developing an SRS is helping the clients understand their own needs. For software systems that are not just automating existing manual systems, requirements have to be visualized and created. Even where the primary goal is to automate an existing manual system, new requirements emerge as the introduction of a software system offers new potential for features, like collecting data, providing new services and performing activities in a different manner, that were either infeasible or not possible without a software system. The basic quality objective of software development is to satisfy the client. The client has to be made aware of these potentials and aided in visualizing and conceptualizing the needs and requirements of his organization. The process of developing an SRS usually helps in this, as it forces the client and the users to think, visualize, interact, and discuss with others to identify the requirements.

Advantages of SRS Document – Following are the main advantages of SRS –

(i) An SRS establishes the basis for agreement on what the software product will do between the client and the supplier.

The basis for agreement is frequently formalized into a legal contract between the client and the developer. So, SRS helps the client clearly describes what it expects from the supplier and the developer clearly understands what capabilities to build in software. Without such an agreement, once the development is over, the project will have an unhappy client, which almost always leads to unhappy developers.

(ii) An SRS provides a reference for validation of the final product. That is, the SRS helps the client determine if the software meets the requirements. With no proper SRS, there is no other way for client to determine if the software being delivered is what was ordered and there is no way the developer can convince the client that all the requirements have been fulfilled.

(iii) A high quality SRS is a prerequisite to high quality software. The quality of SRS has an impact on schedule and cost of the project. Errors can exist in the SRS, the cost of fixing an error increases almost exponentially as time progress. That is, a requirement error, if detected and removed after the system has been developed can cost up to 100 times more than removing it during the requirement phase itself.

Q.39. Why is software requirements specification (SRS) need in a software project. And what are advantages with it. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.38.

Q.40. Explain software requirement specification in detail and why it is necessary. ***(R.G.P.V., June 2016)***

Ans. Refer to Q.33 and Q.38.

Q.41. What are the components of SRS document ?

Ans. Several standards organizations including IEEE have identified the follow nine components that must be addressed when designing and writing SRS.

(i) Interfaces	(ii) Functional capabilities
(iii) Performance levels	(iv) Data structures/elements
(v) Safety	(vi) Reliability
(vii) Security/privacy	(viii) Quality
(ix) Constraints and limitations.	

Q.42. What are the goals of software requirements specification(SR document ?

Ans. There are following goals of SRS document -

(i) Feedback to Customer – SRS document provides a feedba to the customer. It is the customer's assurance that the developme organization understands the issues or problems to be solved and the softwa behaviour necessary to address those problems. Therefore, the SRS shou be written in natural language, in an unambiguous manner that may also inclu charts, tables, data flow diagrams, decision tables etc.

(ii) Problem Decomposition – The simple act of writing dow software requirements in a well-designed format organizes information, plac borders around the problem, solidifies ideas and helps break down the probl into its component parts in an orderly fashion. SRS document decompos the problem into component parts.

(iii) Input to Design Specification – SRS document serves as input to the design specification. The SRS also serves as the parent docum to subsequent documents, such as the software design specification a statement of work. Therefore, the SRS must contain sufficient detail in th functional system requirements so that a design solution can be devised.

(iv) Production Validation Check – SRS document serves as a produ validation check. The SRS also serves as the parent document for testing a validation strategies that will be applied to the requirements for verification.

Q.43. Write any three structures of requirements definition docume (R.G.P.V., Dec. 201

Ans. The software requirements document is also known as the softw requirements specification or SRS. It should include both the user requireme for a system and a detailed specification of the system requirements. Sometim the user and the system requirements may be integrated into a single descripti In other cases, the requirements of the user are defined in an introduction to t system requirements specification. The detailed system requirements may presented as a separate documents when there are a large number of requireme

The IEEE standard suggests the following three requirements documents.

(i) Introduction –

(a) Purpose of the requirements document

(b) Scope of the product

(c) Definitions, acronyms and abbreviations

(d) References

(e) Overview of the remainder of the document.

(ii) General description –

(a) Product perspective	(b) Product functions
(c) User characteristics	(d) General constraints
(e) Assumptions and dependencies.	

(iii) Specific Requirements – These are the requrements which covers functional, non-functional and interface requirements. Obviously, it is the most substantial part of the document but, because of the wide variability in organisational practice, it is difficult to define a standard structure for this section. The requirements can document external interfaces, describe system functionality and performance, specify logical database requirements, design constraints, emergent system properties and quality characteristics.

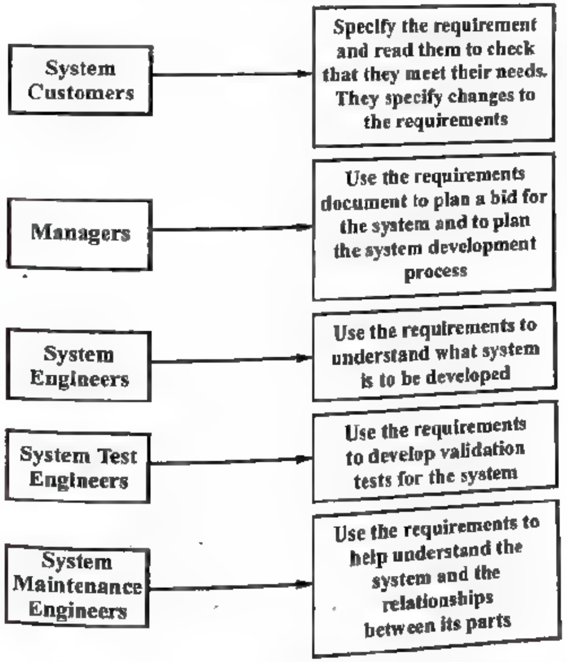

System Customers
Specify the requirement and read them to check that they meet their needs. They specify changes to the requirements
Managers
Use the requirements document to plan a bid for the system and to plan the system development process
System Engineers
Use the requirements to understand what system is to be developed
System Test Engineers
Use the requirements to develop validation tests for the system
System Maintenance Engineers
Use the requirements to help understand the system and the relationships between its parts

Fig. 2.21 Users of a Requirements Document

Q.44. What are the uses of SRS document ?

Ans. The major uses of SRS document are –

(i) Project managers base their plans and estimates of schedu effort and resources on it.

(ii) Development team needs it to develop product.

(iii) The testing group needs it to generate test plans based on t described external behaviour.

(iv) The maintenance and product support staff need it to understa what the software product is supposed to do.

(v) The publications group writes documentation, manuals etc. from

(vi) Customers rely on it to know what product they can expect

(vii) Training personnel can use it to help develop educational materi for software product.

Q.45. What are the important issues which an SRS document mu address ? ***(R.G.P.V., June 2005, 20***

Ans. The basic issues an SRS must address are –

(i) Functionality

(ii) Performance

(iii) Design constraints imposed on an implementation

(iv) External interfaces.

(i) Functional Requirements – They specify which outputs shou be produced from the given inputs. They describe the relationship betwe the input and output of the system. For each functional requirement, a detail description of all the data inputs and their source, the units of measure and t range of valid inputs must be specified.

All the operations to be performed on the input data to obtain the outp should be specified. This includes specifying the validity checks on the input a output data, parameters affected by the operations and equations or other logi operations that must be used to transform the inputs into corresponding outpu For instance, if there is a formula for computing the output, it should be specifie

The system behaviour in abnormal situations is an important part of t specification like invalid input or error during computation. The function requirement must clearly state what the system do if such situations occ Specifically, it should specify the behaviour of the system for invalid inpu and invalid outputs. Furthermore, behaviour for situations where the input i valid even then the normal operation cannot be performed should also b specified. For instance, a situation in an airline reservation system when reservation cannot be made even for valid passengers if the airplane is tota booked. In other words, the system behaviour for all foreseen inputs and foreseen system states should be specified.

(ii) Performance Requirements – Performance requirements of an SRS specify the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. The performance requirements are of two types – static and dynamic.

Static requirements are those that do not impose constraint on the execution characteristics of the system. These include requirements such as the number of terminals to be supported, the number of simultaneous users to be supported, and the number of files that the system has to process and their sizes. These are also known as ***capacity requirements*** of the system.

Dynamic requirements are those that specify constraints on the execution behaviour of the system. These include response time and throughput constraints on the system. Response time is the expected time for the completion of an operation under specified circumstances. Throughput is the expected number of operations that can be carried out in a unit time. For instance, the SRS may specify the number of transactions that must be processed per unit time, or what the response time for a particular command should be. Acceptable ranges of the different performance parameters should be specified, as well as acceptable performance for both normal and peak workload conditions.

(iii) Design Constraints – There are a number of factors in the client's environment that may restrict the choices of a designer. These factors include standards that must be followed, resource limits, operating environment, reliability and security requirements, and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

(a) Standards Compliance – This specifies the requirements for the standards the system must follow. The standards may include the report format and accounting procedures. There can have audit tracing requirements, which require certain kinds of changes, or operations that must be recorded in an audit file.

(b) Hardware Limitations – The software may have to operate on some existing hardware, thus imposing restrictions on the design. Hardware limitations can include the type of machines to be used, operating system available on the system, languages supported, and limits on primary and secondary storage.

(c) Reliability and Fault Tolerance – Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements make the system more complex and expensive. Requirements about system behaviour in the face of certain kinds of faults is specified. Recovery requirements are an integral part here, detailing what the system should do if some failure occurs to ensure certain properties. Reliability requirements are important for critical applications.

(d) Security - Security requirements are important in defe systems and database systems. These requirements impose restrictions o the use of certain commands, control access to data, provide different kin of access requirements for different people, require the use of passwor and cryptography techniques, and maintain a log of activities in the syste

(iv) External Interface Requirements – All the possible interactio of the software with people, hardware, and other software should be clea specified. For the user interface, the characteristics of each user interface o the software product should be specified. A preliminary user manual should created with all user commands, screen formats, an explanation of how t system will appear to the user, and feedback and error messages. The requirements should be precise and verifiable. Hence, a statement like "t system should be user friendly" should be avoided and statements li "commands should be no longer than six characters" or "command nam should reflect the function they perform" used.

For hardware interface requirements, the SRS should specify the logic characteristics of each interface between the software product and the hardw components. If the software is to execute on existing hardware or predetermined hardware, all the characteristics of the hardware, includi memory restrictions, should be specified. In addition, the current use and lo characteristics of the hardware should be given.

The interface requirement should specify the interface with other softw the system will use or that will use the system. This includes the interfa with the operating system and other applications. The format and messa content of each interface should be specified.

Q.46. What are the steps involved in requirement engineering.? ***(R.G.P.V., Dec. 201***

Ans. The requirement engineering process may vary based on t application domain, a few generic steps are common across all types of softw projects. The steps involved in requirement engineering are –

(i) Requirement development

(a) Feasibility study
(b) Requirements elicitatio
(c) Requirement analysis
(d) Requirement specificati
(e) Requirement validation.

(ii) Requirement management.

Q.47. What is the use of requirement engineering ? What are t problems in formulation of requirements ? ***(R.G.P.V., June 20***

Ans. Requirement engineering is the process of establishing the servi that the customer requires from a system and the constraints under whic

system operates and is developed. The requirements themselves are the descriptions of the system services and constraints that are generated during the requirement engineering process. Requirement engineering is a key activity in software and system development. It is essential for major quality attributes, such as system functionality, usuability, and development cost and time. The main task of requirement engineering is to determine and specify explicitly the stated or implied needs, i.e. the requirements a system has to fulfil. These needs have their origin in a certain context, such as stakeholders, goals, assumptions, and the application domain.

Requirements engineering provides the appropriate mechanisms for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a resonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system. The requirements engineering process is described in five different steps – requirements elicitation, requirements analysis and negotiation, requirements specification, system modeling, requirements validation, and requirements management.

Some of the problems that arise during the requirements engineering process are a result of failing to make a clear separation between high level and detailed levels of description. This can be separated by using the term user requirements to mean the high-level abstract requirements and system requirements to mean the detailed description of what the system should do.

Q.48. Why validation is important in the requirement phase ? Justify your answer with proper example. ***(R.G.P.V., June 2015)***

Ans. The development of software begins with the requirements document is ready. One of the objectives of this document is to check whether the delivered software system is acceptable or not. Therefore, it is important that the requirements specification contains no errors and specifies the client's requirements correctly. Also, the longer an error remains undetected, the greater the cost of correcting it. Hence, it is extremely desirable to detect errors in the requirements before the design and development of the software begin.

In validation phase, the work products produced as a consequence of requirements engineering are examined for consistency, omissions, and ambiguity. The basic objective is to ensure that the SRS reflects the actual requirements accurately and clearly. Other objectives of the requirements document are –

(i) To certify that the SRS contains an acceptable description of the system to be implemented.

(ii) To ensure that the actual requirements of the system are reflected in the SRS.

(iii) To check the requirements document for completene accuracy, consistency, requirement conflict, conformance to standards a technical errors.

Requirements validation is similar to requirements analysis as b processes review the gathered requirements. Requirements validation stu the final draft of the requirements document while requirements analysis stu the raw requirements from the system stakeholders or users.

In fig. 2.22, various inputs such as requirements document, organizatio knowledge, and organiza tional standards are shown. The requireme documents should be formulated and organized according to the standards the organizations. The organizational knowledge is used to estimate the real of the requirements of the system. The organi- zational standards are t specified standards follo- wed by the organization according to which t system is to be developed.

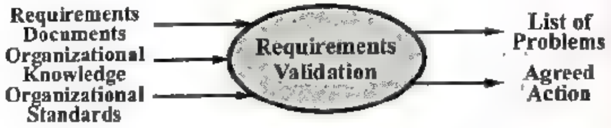


Fig. 2.22 Requirements Validation

The output of requirement validation is a list of problems and agr actions of the problems. The lists of problems indicate the problems encount in the requirements document of the requirement validation process. The ag action is a list that displays the actions to be performed to resolve the probl depicted in the problem list.

Q.49. What are the various requirements validation techniques?

Ans. The requirements validation techniques, which can be used conjunction or individually, are as follows –

(i) ***Requirements Reviews*** – A team of reviewers analyses t requirements systematically.

(ii) ***Prototyping*** – In this technique, an executable model of t system is shown to end-users and customers. They experiment with t model to see if it fulfills their real requirements.

(iii) ***Test-case Generation*** – Requirements should be testable. If t tests for the requirements are conducted as part of the validation process, t frequently reveals requirements problems. If a test is complex or imposs to design, this usually refers that the requirements will be hard to implem and should be reconsidered.

(iv) ***Automated Consistency Analysis*** – If a system model is used to express the requirements in a structured or formal notation then CASE tools may be used to check the consistency of the model. This is shown in fig. 2.23. The CASE tool must construct a requirements database to check the consistency and then, with the help of the rules of the method or notation, check all of the requirements in this database. A requirements analyser provides a report of inconsistencies which it has found.

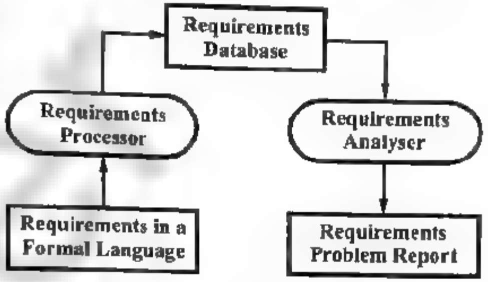


Fig. 2.23 Automated Consistency Checking of Requirements

Q.50. What do you understand by traceability in the context of software requirement specification ? How is traceability achieved ? Why is traceability important considered an important issue ? ***(R.G.P.V., June 2014)***

Or

What do you understand by traceability in the context of software requirements specification ? How is traceability achieved ?

(R.G.P.V., June 2012)

Ans. Traceability means that it would be possible to tell which design component corresponds to which requirement, which code part corresponds to which design component, and which test case corresponds to which requirement, etc. Thus, a given code component can be traced to the corresponding design component, and a design component can be traced to a specific requirement that it implements and vice versa. Traceability analysis is an important concept and is frequently used during software development. For example, by doing a traceability analysis, we can tell whether all the requirements have been satisfactorily addressed in all phases. It can also be used to assess the impact of a requirements change. That is, traceability makes it easy to identify which parts of the design and code would be affected when certain requirement change occurs. It can also be used to study the impact of a bug on various requirements, etc.

Traceability techniques facilitate the impact of analysis on changes of the project, which is under development. Traceability information is stored in a traceability matrix, which relates requirements to stakeholders or design module. The traceability, matrix refers to a table that correlates high-level requirements with the detailed requirements of the product. Mainly, five types of traceability tables are maintained. These are listed in table 2.1.

Table 2.1 Types of Traceability Tables

S.No.	Traceability Table	Description
(i)	Features traceability	Indicates how requirements relate to importa features specified by the user.
(ii)	Source traceability	Identifies the source of each requirement b linking the requirements to the stakeholders wh proposed them. When a change is propos information from this table can be used to fi and consult the stakeholders.
(iii)	Requirement traceability	Indicates how dependent requirements in th SRS are related to one another. Information fro this table can be used to evaluate the numbe of requirements that will be affected due to th proposed change(s).
(iv)	Design traceability	Links the requirements to the design modul where these requirements are implemented Information from this table can be used to evalua the impact of proposed requirements changes o the software design and implementation.
(v)	Interface traceability	Indicates how requirements are related to intern interface and external interface of a system.

In a traceability matrix each requirement is entered in a row and colum of the matrix. The dependencies between different requirements are represented in the cell at a row and column intersection. In fig. 2.24, 'U' in the row and column intersection indicates the dependencies of the requirements in the row on the column and 'R' in the row and column intersection indicates the existence of some other weaker relationship between the requirements.

Req. ID	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

Fig. 2.24 Traceability Matrix

A traceability matrix is useful when less number of requirements are to b managed. However, traceability matrices are expensive to maintain when large system with large require-ment is to be developed. This is because larg requirements are not easy to manage. Hence, the traceability information o large system is stored in the 'requirement database' where each requirement explicitly linked to related requirements. This helps to assess how a change one requirement affects the different aspects of the system to be develop

UNIT 3

SOFTWARE DESIGN

THE SOFTWARE DESIGN PROCESS, DESIGN CONCEPTS AND PRINCIPLES

Q.1. What is design ? Discuss the objectives of software design.
(R.G.P.V., Dec. 2006, June 2008, Dec. 2014, June 2016)

Ans. The term design is used in two ways. While used as verb, it means – the design process and used as noun, it means – the result of design process. Design is a meaningful engineering representation of something that is to be built. The result of a design process is called as ***design model*** or the ***design of the system.***

The design for computer software is as important as blueprint for a house, else the result will be chaos. The design of a system is a plan for a solution for the system. Here a system is considered to be a set of components having clearly defined behaviour which interacts with one another in a predefined manner to produce some behaviour or services for its environment.

The software design process often has two levels. One of which decides the modules for the system, their specifications and their interconnections. This is known as the ***system design*** or ***top-level design.*** The other one, known as ***detailed design*** or ***logic design,*** decides the internal design of the modules.

A ***design-methodology*** is a systematic way of creating a design by applying a set of techniques and guidelines. Most design methodologies concentrate on the system design. Also most recent ones, offer a set of guidelines to help the developer, designing a system.

Objectives of Software Design – Mcglaughlin suggests three characteristics that serve as a guide for the evaluation of a good design –

(i) All of the explicit requirements contained in the analysis model must be implemented by the design. And, the design must accommodate all of the implicit requirements desired by the customer.

(ii) The design must be a readable, understandable guide for those who produce code and for those who test and subsequently help the software.

(iii) A complete picture of the software should be provided by t design, addressing the data, functional and behavioural domains from t implementation perspective.

Each of these characteristics refers to a goal of the design process.

Q.2. What is a design ? Describe the difference between conceptu design and technical design. ***(R.G.P.V., Dec. 201***

Ans. Refer to Q.1.

For a candidate system conceptual design describes the inputs, outpu databases and procedures that meets the user's requirement. Followi conceptual design is technical design. This produces the working system b defining the design specifications that tell programmers exactly what t candidate system must do.

Q.3. Differentiate between good design and bad design.

Ans. The comparison of good and bad design against its key characteristi is provided in table 3.1.

Table 3.1

S.No.	***Characteristics***	***Good Design***	***Bad Design***
(i)	Change	Change in one part of the system does not always require a change in another part of the system.	One conceptual change requires changes to many parts of the system
(ii)	Logic	Every piece of logic has one and one home.	Logic has to be duplicated.
(iii)	Nature	Simple	Complex
(iv)	Cost	Small	Very high
(v)	Link	The logic link can easily be found.	The logic link cannot be remembered.
(vi)	Extension	System can be extended with changes in only one place.	System cannot be extended so easily

Q.4. What do you understand by design activities ?

Ans. A good software design is seldom arrived by using a single st procedure but rather by iterating over a series of steps called the design activiti Depending on the order in which various design activities are performed, w can broadly classify design activities into two important stages – high lev design and detailed design. The meaning and scope of these two stages c vary considerably from one design methodology to another.

Q.5. Differentiate between structure chart and flow chart as design representation techniques. ***(R.G.P.V., Dec. 2010)***

Ans. The flow chart is a convenient technique to represent the flow of control in a system. A structure chart differs from a flow chart in three principal ways –

(i) It is usually difficult to identify the different modules of the software from its flow chart representation.

(ii) Data interchange among different modules is not represented in a flow chart.

(iii) Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.

Q.6. Explain software design process. What are the technical criteria of a good design ?

Or

Write short note on software design. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. Software design is an iterative process of translating requirements into a plan for software construction. Initially, the plan is represented at a high level of abstraction. As design iterations take place, subsequent refinement results in design representations at much lower abstraction level.

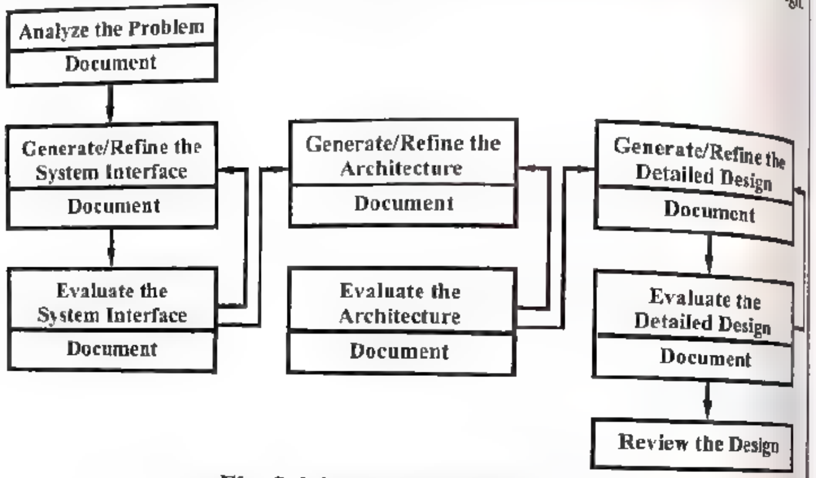
The software design process can be decomposed mainly into the following three levels or phases of design –

(i) Interface Design – Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e. during interface design, the internals of the system are completely ignored in favor of the relationship between the system and its environment – the system is treated as a black box.

Attention is focused on the dialogue between the target system and the users, devices and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices, which are collectively called agents.

(ii) Architectural Design – It is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design the overall structure of the system is chosen, but the internal details of major components are ignored. Architectural design adds important details ignored during interface design. Design of the internals of the major components is ignored until the last phase of design.

(iii) Detailed Design – It is the specification of the internal eleme of all major system components, their structure, properties, relationshi processing, and often their algorithms and data structures. Detailed desi fills in all the design specifications still left open after architectural design.



Analyze the Problem
Document
Generate/Refine the System Interface
Document
Evaluate the System Interface
Document
Generate/Refine the Architecture
Document
Evaluate the Architecture
Document
Generate/Refine the Detailed Design
Document
Evaluate the Detailed Design
Document
Review the Design

Fig. 3.1 Software Design Process

For the evaluation of the quality of the design representation, follow are the technical criteria for good design –

(i) A design should have an architectural structure that has t following characteristics –

(a) It has been created by recognizable design patterns.

(b) It is having components that show good desi characteristics.

(c) It can easily be applied in an evolutionary manner, th facilitating implementation and testing.

(ii) A design should be modular i.e., the software should be divid logically into several elements performing specific functions.

(iii) A design should have different representations of da architecture, interfaces, and components.

(iv) It should be such that it leads to data structures, suitable for t objects to be implemented and drawn from recognizable data patterns

(v) It should lead to components having independent functio characteristics.

(vi) It should lead to interfaces that decrease the connectio complexity between modules and with external environment.

(vii) It should be derived with the help of a repeatable method dri by information collected during software requirements analysis.

Q.7. Define software design process. Describe various steps involve in software design.

(R.G.P.V., June 2015)

Ans. **Software Design Process** – Refer to Q.6.

Steps in Software Design – The steps involved in the general design process are shown in fig. 3.2 and described below –

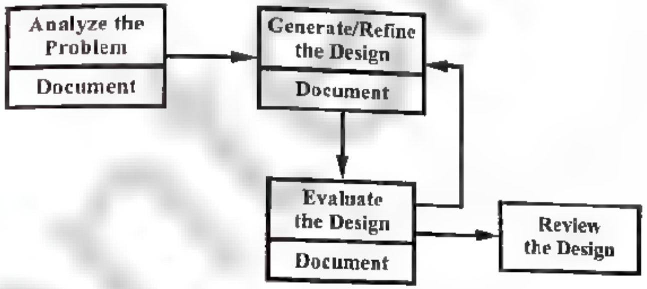


Fig. 3.2 General Design Process

(i) Analyze the Problem – This step should produce a clear design problem statement to guide the remainder of the process.

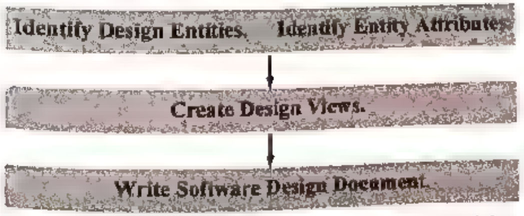
(ii) Generate/Refine the Design – The next step is to generate design ideas, including design alternatives, or to refine ideas generated and evaluated before. These ideas are recorded in various design documents.

(iii) Evaluate the Design – The newly generated or refined design ideas are evaluated, especially against the problem statement.

(iv) Review the Design – The final step in the process is a review by people. Ideally, reviewers are drawn from outside the design group, and may include clients, upper management representative of other parts of the organization, or design consultants. The goal is to ensure that the design succeeds in solving the design problem.

Q.8. What are the various activities involved in the software design process ?

Ans. The software design process consists of the activities as shown in fig. 3.3.



Identify Design Entities. Identify Entity Attributes
Create Design Views.
Write Software Design Document.

Fig. 3.3 Software Design Process Activities

(i) Identification of Design Entities – A design entity is an elem or component of design, which is structurally different from other elemen A design entity is referenced with a unique identity throughout the design

(ii) Identification of Entity Attributes – Entity attributes are named characteristics or properties of a design entity. An attribute provides statement of existence of the entity in the system. A set of attributes of design entity defines the existence of the design and its importance and visi in the system.

(iii) Creating Design Views – A design view is a subset of informati of attributes of a design entity that is specifically suited to the needs of activity of the software. It means that the design view provides a view existence of a set of design entities in the software.

(iv) Writing Software Design Description – Software desi description is a representation of a software system created to facilitate analy planning, implementation, and decision-making. It is a blueprint or prototy model of the software system. The document containing the software des description is known as software design description document or sim software design document.

Q.9. Explain software design process. Differentiate between sys design and detailed design. ***(R.G.P.V., Dec. 20***

Ans. **Software Design Process** – Refer to Q.6.

System design, which is also known as top-down design, aims to iden the modules that should be in the system, the specification of these modu and how these modules interact with each other to produce the desired resu Finally, at the end of the system design all the major data structures, formats, output formats, and the main modules in the system and th specifications are decided. The main focus in the system design is on identif the modules. In other words, we can say, the attention is on what compon are needed. Most methodologies focus on system design.

In detailed design, which is sometimes also called bottom-up design, internal logic of each of the modules specified in system design is deci Again the details of the data structures and algorithmic design of each of modules is specified in this phase. The logic of a module is usually specified a high level design description language, which is independent of the ta language in which the software will eventually be implemented. During deta design the focus is on designing the logic for each of the modules In deta design, the attention is on how the components can be implemented in softw is the issue.

Q.10. Write a detailed note on software design and its tools. ***(R.G.P.V., June 2007)***

Ans. **Software Design** – Refer to Q.6.

Tools – An approach to software design is suggested by 'structured methods' which are sets of notations and guidelines for software design. Examples of structured methods are structured design, structured systems analysis, Jackson system development, and various approaches to object-oriented design.

The use of structured methods include producing graphical system models and provides large amounts of design documentation. CASE tools are used to support particular methods. Structured methods have been used successfully in several large projects. They can provide significant cost reductions as they use standard notations and guarantee that standard design documentation is produced.

A structured method includes a design process model, notations to represent the design, report formats, rules, and design guidelines. Structured methods may help some or all of the following models of a system –

(i) A data-flow model where the system is modelled using the data transformation which happen as it is processed.

(ii) An entity-relation model which is used to describe the basic entities in the design and the relations between them.

(iii) A structural model where the system components and their interactions are documented.

(iv) Object-oriented methods are an inheritance model of the system, models of the static and dynamic relationships between objects and a model of how objects interact with each other when the system is executing.

Q.11. Differentiate between analysis and design. (R.G.P.V., June 2007)

Ans. The software engineer and the customer perform an active role in software requirements engineering – a set of activities that is known as ***analysis.*** The customer tries to reformulate a vague system-level description of data, function and behaviour into concrete detail. The developer works as interrogator, consultant, problem solver, and negotiator.

Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design. Requirements engineering activities provide the specification of software's operational characteristics, specify software's interface with other system elements, and determine constraints that software must satisfy. Requirement analysis gives the software designer with a representation of information, function, and behaviour that can be translated to data, architectural interface, and component-level designs. Finally the requirements specification gives t developer and the customer with the means to ascertain quality after softw development is built.

Design is an approach to represent something that is to be built. It can traced to a customer's requirements and at the same time ascertained f quality against a set of predefined criteria for "good" design. In the softwa engineering context, design concentrates on four major areas of concern data, architecture, interfaces and components. Software engineers desi computer based systems, but the skills needed at each level of design wo are different. At the data and architectural level, design concentrates on patter because they apply to the application to be built. At the interface level, hum ergonomics often tell design approach. At the component level, a "programmi approach" results in effective data and procedural designs. Computer softw is complicated, so we require a blueprint the design. Design starts with t requirements model. This model is transformed into four levels of desig detail – the data structure, the system architecture, the interface representati and the component level detail. During each design activity, we use bas concepts and principles that provide high quality ultimately, a design specificati is produced.

Q.12. Briefly describe software analysis and design tools.

(R.G.P.V., Dec. 201

Ans. Analysis and design tools enable a software engineer to create mode of the system to be built. The models contain a representation of data, functi and behaviour and characterization of data, architectural component level an interface design. By performing consistency and validity checking on th models, analysis and design tools provide a software engineer with some degr of insight into the analysis representation and help to eliminate errors befo they propagate into the design, or worse, into the implementation itself. Som tools are given below –

(i) Data Flow Diagram (DFD) – The first step is to draw a da flow diagram (DFD). The DFD was first developed by Larry Constantine a way of expressing system requirements in a graphical form. This led to modular design.

A DFD, also known as a "bubble chart" has the purpose of clarifyi system requirements and identifying major transformations that will beco programs in system design.

(ii) Data Dictionary – The repository of all data definitions for t organizational applications.

(iii) Structured English – Modified form of the English language used to specify the logic of information system processes. Although there is no single standard structured English typically relies on action verbs and noun phrases and contains no adjectives or adverbs.

(iv) Decision Tree – A graphical representation of a decision situation in which decision points (nodes) are connected together by arcs (one for each alternative on a decision) and terminate in ovals (the action which is the result of all the decisions made on the path that leads to that oval).

(v) Decision Table – A matrix representation of the logic of a decision, which specifies the possible conditions for the decision and the resulting actions.

(vi) Structure Charts – It is derived from DFD and represent the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail. It is a hierarchical structure of modules.

(vii) HIPO Diagram – Hierarchical input process output diagram is a combination of two organized method to analyze the system and provide the means of documentation. Analyst uses HIPO diagram in order to obtain high level view of system function. These diagrams are good for documentation purpose. Their graphical representation makes it easier for engineer and managers to get the pictorial idea of the system structure.

(viii) ER Model – It is a type of database model based on the nation of real world entities and relationship among them. We can map real world scenario onto ER database model. ER model creates a set of entities with their attributes, a set of constraints and relation among them.

Q.13. Discuss the various design concepts.(R.G.P.V., June 2003, Dec. 2009)

Or

Explain about the various design concepts considered during design.

(R.G.P.V., May 2019)

Ans. A set of fundamental design concepts has evolved. They are –

(i) Abstraction – Abstraction is one of the fundamental ways that we as humans cope with complexity. Many levels of abstraction can be posed at the time of considering a modular solution to any problem. A solution is stated in broad terms using the language of the problem environment at the highest level of abstraction. A more procedural orientation is taken at lower levels of abstraction.

(ii) Refinement – Actually, refinement is a process of elaboration. A macroscopic statement of function is decomposed in a stepwise fashion to develop a hierarchy until programming language statements are reached. One or several instructions of the given program are decomposed into more detai instructions in each step. The concepts of abstraction and refinement complementary.

(iii) Modularity – Software is partitioned into separately named a addressable components, called modules, that are integrated to meet prob requirements. It has been said that "modularity is the single attribute of softw that permits a program to be intellectually manageable".

(iv) Software Architecture – It refers to the overall structure of software and the ways in which that structure provides conceptual integrity a system. A software architecture is the development work product that g the highest return on investment with respect to quality, schedule, and cost.

(v) Control Hierarchy – It is also known as program structure represents the organization of program components and implies a hierarchy control. The most common notation to represent control hierarchy is the tr diagram.

(vi) Structural Partitioning – The program structure can be divi both horizontally and vertically. For each major program function, horizon divides define separate branches of the modular hierarchy. Control modu are used to coordinate communication between and execution of the functio Vertical partitioning also called factoring, implies that control and work sho be distributed top-down in the program structure.

(vii) Data Structure – It represents the logical relationship amo individual elements of data. Data structure states the organization, methods access, degree of associativity, and processing alternatives for informatio

(viii) Software Procedure – It concentrates on the processing det of each module individually. Procedure must give an exact specification processing including sequence of events, exact decision points, repeti operations and even data organization and structure.

(ix) Information Hiding – It suggests that modules "characterized by design decisions that hides from all others." It means th modules should be specified and designed so that information present with module is not accessible to other modules that do not need for such informati

Q.14. Explain software design principles.

(R.G.P.V., Dec. 2005, June 20

Ans. Following are the basic design principles that enable the softw engineer to navigate the design process –

(i) There should be no tunnel vision in the design process. A g software designer should make use of alternative approaches, judging according to the problem requirements, the resources available for the job, and the design concepts.

(ii) There should be traceable design to the analysis model. It is essential to have a means for tracking how requirements have been met the design model, since a single design element of the model often traces to multiple requirements.

(iii) The design patterns should not be reinvented. Since systems are constructed with the help of a set of design patterns, many of them are likely been found before. The design patterns should always be selected as an alternative to reinvention.

(iv) The intellectual distance between the software and the problem should be minimized as it exists in the real world. That is, the software design structure should mimic the problem domain structure (whenever possible).

(v) There should be uniformity and integrity in the software design. Uniformity comes from its appearance, which should be such that it appears to be developed entirely by one person. Integrity comes by carefully defining the interfaces between its components.

(vi) The design should be structured to make possible change. The design concepts enable a design to achieve this principle.

(vii) Even when to aberrant data, events, or operating conditions are encountered, the design should be structured to degrade gently. Well-designed software should never "bomb". It should be designed to accommodate unusual situations, and if it must finish processing do so in a graceful way.

(viii) Coding is not design and design is not coding. Even having detailed procedural designs for the program components, the design model has higher level of abstraction than that of a source code. The only design decisions made at the coding level address the small implementation details that make possible the procedural design to be coded.

(ix) The design should be assessed for quality because it is being created, not after the fact. A number of design concepts and design measures are available to help the designer in assessing quality.

(x) There should be a review of design to minimize conceptual (semantic) errors. Sometimes there is a tendency to concentrate on very small details when the design is reviewed, missing the forest for the trees. A design team should guarantee that key conceptual elements of the design have been addressed before worrying about the syntax of the design model.

Q.15. What is software design ? Explain various principles and design concepts of software design.

(R.G.P.V., May 2018)

Ans. Refer to Q.6, Q.14 and Q.13.

Q.16. What do you understand by design review ? What kinds of mistakes are normally pointed out by the reviewers ? ***(R.G.P.V., June 201***

Ans. Software design reviews can be defined as the well-documented, comprehensive, and systematic examinations of a design. These design reviews are used to assess the adequacy of the design requirements, to assess the capability of the design to meet these requirements, and to recognize problems.

Software design review is defined by the IEEE as 'a formal meeting at which a system's preliminary or detailed design is presented to the user, customer, or other interested parties for comment and approval'.

These reviews are taken place at the end of the design phase to resolve issues related to software-related design decisions, architectural design, and component-level and interface design of the entire software or a part of it such as a database. Examination of development plans, requirements specifications, design specifications, testing plans and procedures, all other documents and activities associated with the project should also be included in the software design reviews.

SOFTWARE MODELING AND UML, ARCHITECTURAL DESIGN, ARCHITECTURAL VIEWS AND STYLES, USER INTERFACE DESIGN, FUNCTION-ORIENTED DESIGN, SA/SD COMPONENT BASED DESIGN, DESIGN METRICS

Q.17. Write a short note on software modeling. ***(R.G.P.V., June 2014)***

Ans. A model can be called as a simplified version of a real system. It can be thought of as a model capturing aspects important for some applications while omitting the rest. When the size of a problem increases, the perceived complexity increases exponentially due to human cognitive limitations. Therefore, to develop a good understanding of any problem, it is necessary to construct a model of the problem. Modeling has turned out to be a very essential tool in software design and helps to effectively handle complexity in a problem.

Different kinds of models are obtained on the basis of the aspects of the actual system are ignored while constructing the model. To understand this let us consider the models constructed by an architect of a large building. While constructing the frontal view of large building, the architect ignores aspects like floor plan strength of the wall, details of inside architecture, etc. while constructing the floor plan, he completely ignores the frontal view, thermal and lighting characteristics, etc. of the building.

A model in the context of software development can be graphical, textual, mathematical or program code-based. Graphical models are very famous because they are simple to understand and construct. UML is mainly a graphical modeling tool. However, UML often needs separate textual explanations to accompany the graphical models.

Q.18. Write a short note on UML. ***(R.G.P.V., June 2014)***

Ans. Unified Modeling Language (UML) is a language used to create an abstract system scenario, by visualizing, specifying, constructing, and documenting various parts and components of the system into a representative model enabling software system development. UML has its syntax and semantics. It provides a set of notations, such as rectangles, lines, ellipses, etc., to create models of systems that would be useful in documenting the design and analysis results.

The main goals in the design of UML are as follows –

(i) Offer users with a ready-to-use, expressive, and visual modeling language to create models.

(ii) Offer a language and notations to enhance concepts to higher order representation.

(iii) Do not depend on OO languages.

(iv) Assist higher-level development concepts such as component technology, rapid application development, reusability, interoperability, and portability.

The following benefits are provided by the UML-based modeling –

(i) Enhances communications among project teams.

(ii) Enhances the developer's insight and visualization of the complex system.

(iii) Developers learn faster to include the system's intricacies properly in the design.

(iv) Prototype design is more suitable, where the specific complexity of structure and behaviour is taken into account in each iteration. This enhances the system in increments, and part by part.

Q.19. What are the different system views that can be modelled using UML ? What are the different UML diagrams which can be used to capture each of the views ?

Or

Explain the use of UML for object-oriented design.
(R.G.P.V., June 2010, 2011)

Or

What are the different system views that can be modelled using UML?
(R.G.P.V., June 2015)

Ans. In UML, a system is represented using following views (models) to describe the system from distinctly different perspectives –

(i) User Model View – This view defines the functionalities provided by the system to its users.

*(ii) **Structural Model View*** – This view represents the structure of the problem in terms of the types of objects required to understand the working of a system and to its implementation.

*(iii) **Behavioural Model View*** – This view represents the interactions between various objects to realize the system behaviour.

*(iv) **Implementation Model View*** – This view represents the structural and behavioural aspects of the system because they are to be built.

*(v) **Environment Model View*** – This view represents the structural and behavioural aspects of the environment in which the system is to be implemented.

Fig. 3.4 shows the different UML diagrams which can be used to capture each of the views.

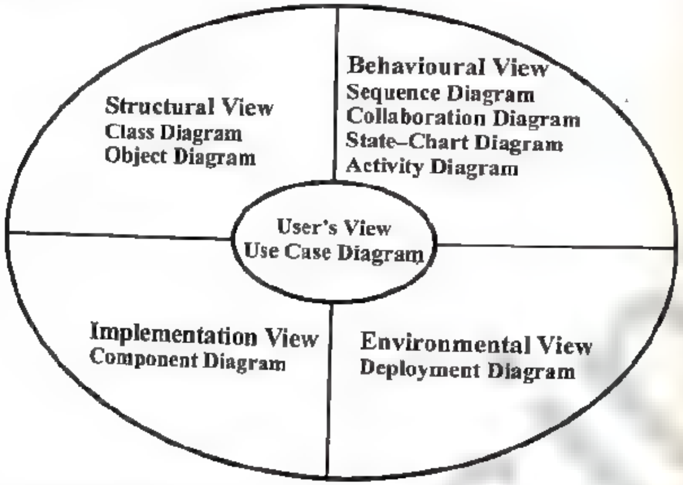


Fig. 3.4 Different Types of Diagrams and Views Supported in UML

Q.20. Write a note on architectural design. ***(R.G.P.V., Dec. 2017)***

Ans. IEEE defines architectural design as the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

Architectural design represents the structure of data and program components that are required to build a computer-based system. It considers the architectural style that the system will take, the structure, and properties of the components that constitute the system and the inter-relationships that occur among all architectural components of a system.

Q.21. Explain architectural and procedural design for a software. ***(R.G.P.V., May 2019)***

***Ans.* Architectural Design** – Refer to Q.20.

Procedural Design – The objective in procedural design is to transform structural components into a procedural description of the software. Information obtained from the process and control specifications and the state transition diagrams serve as a basis for procedural design this steps occurs after the data and program structures have been established, i.e. after architecture design. Procedural details can be represented in different ways –

(i) Graphical design notation

(ii) Tabular design notation

(iii) Program design language (PDL).

Q.22. What do you understand by software architecture ? What are framework models ? ***(R.G.P.V., June 2006)***

Ans. Software architecture represents the overall software structure and the ways in which that structure offers conceptual system integrity for a system. Generally, architecture is the hierarchical program components.

Bass, Clements, and Kazman define software architecture in the following way –

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

One aim of software design process is to derive an architectural rendering of a system which provides a framework from which more detailed design activities are performed.

Following are the set of properties that should be specified as part of an architectural design –

*(i) **Structural Properties*** – These are the properties which define the system components and the way those components are packaged and interact.

*(ii) **Extra-functional Properties*** – These properties of an architectural design address how the design architecture achieves requirements for performance, reliability, adaptability, capacity, security, and other system characteristics.

*(iii) **Families of Related Systems*** – The architectural design should draw upon repeatable patterns found in the design of families of related systems. In short, the design should be able to reuse architectural building blocks.

There are five different types of models used to represent the architectural design. These are – ***structural models***, which represent architecture as a well organized collection of program components; ***framework models***, which enhance the design abstraction level by trying to identify repeated patterns found in similar applications; ***dynamic models***, which address the behavioural view of the program architecture, thus indicating how the structure or system configuration may change externally; ***process models***, which emphasize on the business design or the design of technical process that the system must accommodate; and finally ***functional models***, which can be used to represent the functional hierarchy of a system. Various ***architectural description languages (ADLs)*** have been developed to represent these models.

Q.23. Discuss the architecture design process. (R.G.P.V., June 2004)

Ans. The architectural design process is considered as developing a basic structural framework for a system. It includes determining the major system components and their communications. The following are the three advantages of explicitly designing and documenting a software architecture –

(i) Stakeholder Communication – A high-level system presentation of system is the architecture. It may be used as a focus for discussion by a variety of different stakeholders.

(ii) System Analysis – To make the system architecture explicit at the initial stage of system development implies that some analysis may be performed.

(iii) Large-scale Reuse – The transfer of the architecture can be done across system having same requirements and hence can assist large-scale software reuse. There may be possibility of developing product-line architectures where the same architecture is used across several related systems.

The following activities are common to all architectural design processes –

(i) System Structuring – The system is structured into several principal sub-systems, which are independent software units. Communications between sub-systems are recognized.

(ii) Control Modeling – A general control relationships model is developed between the parts of the system.

(iii) Modular Decomposition – Each identified sub-system is decomposed into modules. The architect must decide on the module types and the types of their interconnections.

These activities are usually interleaved instead of conducting in sequence.

The result of the architectural design process is an architectural design document. It consists of several graphical representations of the system models along with related descriptive text. It should describe how the system is structured into subsystems and how each subsystem is structured into modules.

The different graphical models of the system present different perspectives on the architecture. The following architectural models may be developed –

(i) A static structural model that depicts the subsystems or components that are to be developed as separate units.

(ii) A dynamic process model that depicts how the system is organized into processes at run-time. This may be different from the static model.

(iii) An interface model that establishes the services provided by each subsystem through their public interface.

(iv) Relationship models that depict relationships like data flow between the subsystems.

Q.24. What do you mean by an architectural style ?

Ans. Architectural styles define a family of systems in terms of a pattern of structural organization. They also characterize a family of systems that are related by sharing structural and semantic properties. In essence, the purpose of using architectural styles is to develop a structure for all the components present in a system. If an existing architecture is to be reengineered, then imposition of an architectural style results in fundamental changes in the structure of the system. Also, this change includes reassignment of the functionality performed by the components.

Q.25. How do you assess an architectural style that has been derived ? (R.G.P.V., June 2017)

Ans. The assessment of an architectural style that has been derived in the following ways –

Data – How data communication between components take place ? Is data flow continuous or discrete ? What is data transfer mode (i.e. either one-to-one or globally available) ? What is the role of data components, if exist ? How data and function components interact with each other ? Does the data component interact to other components actively or passively ?

Control – Within architecture, how control management take place ? Within control hierarchy (if exist), what is the role of components ? Within the system, how the control transfer take place between components ? How control sharing is done among components ? What is the topology that control defines ? Is there a synchronization of control ? Within a system, how interaction of control and data takes place ?

These questions gives the early assessment of architectural style.

Q.26. Discuss the different classification of architectural styles with respect to software and discuss each style in detail. (R.G.P.V., June 2012)

Ans. The various architectural styles are as follows –

(i) Data-centered – In this architectural style, a data store sits at the center and is accessed frequently by other components which add, delete, update, or modify data within the store. Fig. 3.5 shows a typical data-centered style.

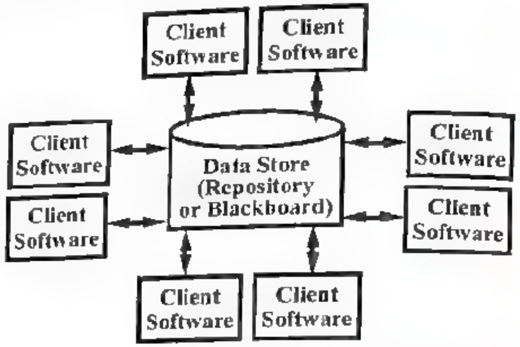
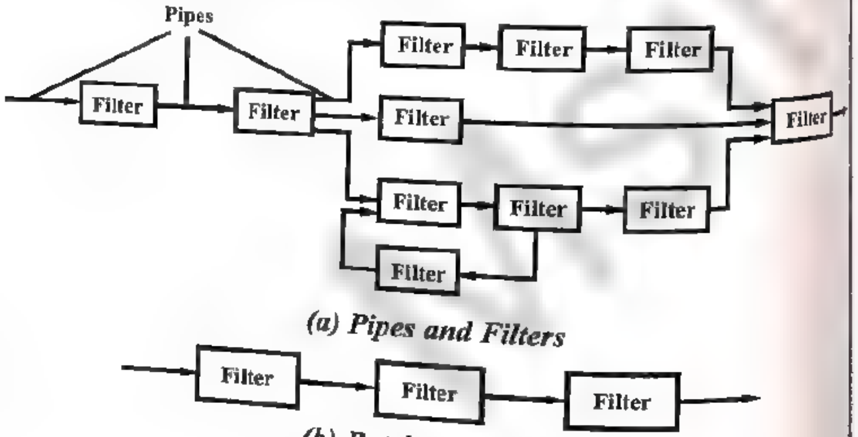


Fig. 3.5 Data-centered Architecture

Observe the fig. 3.5, where client software accesses a central reposit (i.e., data store). The data store can be passive in some cases i.e., cli software accesses the data independent of any change or the actions of oth client software.

Data-centered architectures enhance integrability i.e., currently pres components can be changed and new client components can be added to th architecture, regardless of other clients. Also, the blackboard mechanism i used to pass data among clients. The processes are executed independently b client components.

(ii) Data-flow – This architecture is used in case of th transformation of input data into output data through a series of computation or manipulative components. Fig. 3.6 shows these architectures.



(a) Pipes and Filters

(b) Batch Sequential

Fig. 3.6 Data-flow Architectures

In fig. 3.6 (a), a pipe and filter pattern has a set of components call ***filter***. These filters are connected by pipes that transmit data from o

component to the next. Each filter is designed to accept data input of a certain form and produces data output of a specified form, which goes to the next filter as its input. No two neighbouring filters require to know the working of each other.

Fig. 3.6 (b) illustrates a batch sequential architecture, where the data flow degenerates into a single line of transforms. This pattern takes a batch of data and then a series of sequential components i.e., filters are applied to transform it.

(iii) Call and Return – This architectural style helps a software designer to get a program structure that is easier to modify and scale. The following are the substyles of this category –

(a) Main Program/Subprogram Architecture – It decomposes function into a control hierarchy where a main program calls various program components, which in turn may call still other components. This type of architecture is shown in fig. 3.7.

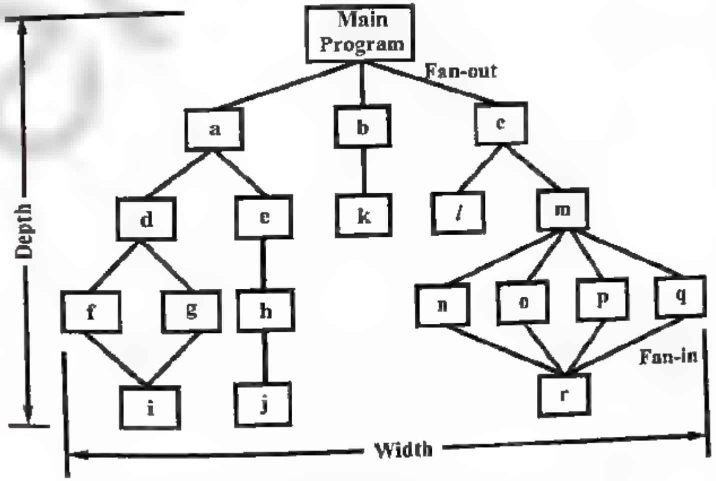


Fig. 3.7 Structure Terminology for a Call and Return Architectural Style

(b) Remote Procedure Call Architecture – In this, components of a main program or subprogram architecture are distributed over a network across several computers.

(iv) Object-oriented – The system components encapsulate data and the required operations for data manipulation. Communication and coordination is established via message passing between components.

(v) Layered – Fig. 3.8 shows the basic structure of a layered architecture.

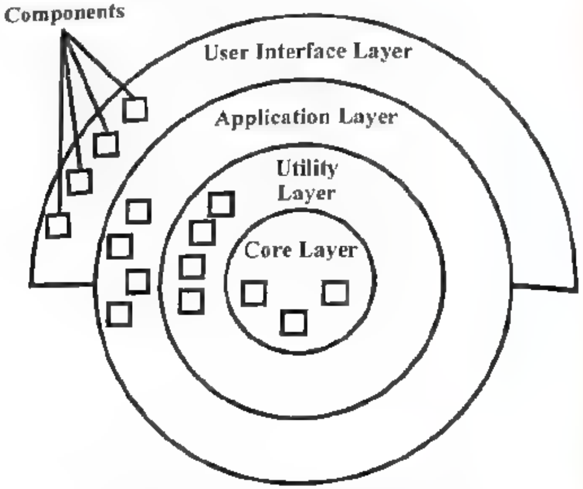


Fig. 3.8 Layered Architecture

This architecture consists of a number of different layers. Each layer performs operations that progressively become closer to the machine instruction set. At the outermost layer, called ***user interface layer***, components service user interface operations. At the innermost layer, known as ***core layer***, components perform operating system interfacing. The two intermediate layers i.e., ***application layer*** and ***utility layer***, offer application software functions and utility services, respectively.

Q.27. Explain why it may be necessary to design the system architecture before the specification is written. ***(R.G.P.V., Dec. 2015)***

Ans. The architecture may have to be designed before specifications are written to provide a means of structuring the specification and developing different subsystems specifications concurrently, to allow manufacture of hardware by sub-contractors and to provide a model for system costing.

Writing specification for the whole system might bring great complexity and it is difficult to formulate it. Therefore, it is easier to divide the system into simpler subsystems and define their specification and it will save you the hassle of defining specification and put it into the respective subsystem. Hence we can concurrently develop subsystems and the specifications to be readily into the implementation stage.

Q.28. Differentiate between data flow model and control flow model. ***(R.G.P.V., June 2008, 2015)***

Ans. In a data-flow model, functional transformations process their inputs and produce outputs. Data flows from one to another and is transformed as it moves through the sequence. Each processing step is implemented as a transform. Input data flow through these transforms until converted to output.

On the other hand, control-flow models are used as a basis for representing the transformation of control. Control models include centralised control and event models. In centralised models, control decisions are made depending on the system state, in event models external event control the system.

Q.29. What is user interface design ? How much it is useful in system design ? ***(R.G.P.V., Dec. 2008)***

Or

Write a short note on user interface design. ***(R.G.P.V., June 2014)***

Ans. User interface design is the design of the interface between a human and the computer. It creates an effective communication medium between a human and a computer. User interface design begins with the identification of user, task, and environmental requirements.

Useness in System Design – Good user interface design is critical to the success of a system. An interface that is difficult to use will, at best, result in a high level of user errors. At worst, users will simply refuse to use the software system irrespective of its functionality. If information is presented in a confusing or misleading way, users may misunderstand the meaning of information. They may initiate a sequence of actions that corrupt data or even cause catastrophic system failure.

Q.30. Describe the golden rules for interface design. (R.G.P.V., May 2019)

Ans. The golden rules divided into three groups which are as follows –

(i) Place users in control

(ii) Reduce users memory load

(iii) Make the interface consistent.

Each group contains a number of specific rules. The rules for each group are as follows –

(i) Place users in Control –

(a) Use modes judiciously (modeless)

(b) Allow users to use either the keyboard or mouse (flexible)

(c) Allow users to change focus (interruptible)

(d) Display descriptive messages and text (helpful)

(e) Provide immediate and reversible actions, and feedback (forgiving)

(f) Provide meaningful paths and exits (navigable)

(g) Accommodate users with different skill levels (accessible)

(h) Make the user interface transparent (facilitative)

(i) Allow users to customize the interface (preferences)

(j) Allow users to directly manipulate interface objects (interactive)

(ii) Reduce Users' Memory Load –

(a) Relieve short-term memory (remember)

(b) Rely on recognition, not recall (recognition)

(c) Provide visual cues (inform)

(d) Provide defaults, undo, and redo (forgiving)

(e) Provide interface shortcuts (frequency)

(f) Promote an object-action syntax (intuitive)

(g) Use real-world metaphors (transfer)

(h) User progressive disclosure (context)

(i) Promote visual clarity (organize)

(iii) Make the Interface Consistent –

(a) Sustain the context of users' tasks (continuity)

(b) Maintain consistency within and across products (experience)

(c) Keep interaction results the same (expectations)

(d) Provide aesthetic appeal and integrity (attitude)

(e) Encourage exploration (predictable).

Q.31. Discuss the interface design guidelines that will result in good interface. ***(R.G.P.V., June 2006)***

Or

What are the fundamental principles of user interface design ? Explain and enumerate the end user requirements in user interface design. ***(R.G.P.V., June 2011, 2015)***

Ans. The designers of user interface must consider the physical and mental capabilities of the software users. Humans have a limited short-term memory thus they make mistakes in case of handling too much information or under stress. They have a diverse range of physical capabilities. Designer must take all of these into account when designing user interfaces.

Table 3.2 User Interface Design Principles

S.No.	***Principle***	***Description***
(i)	User familiarity	The interface should use terms and concepts which are drawn from the experience of the people using the system.
(ii)	Consistency	The interface should be consistent in that comparable operations should be activated in the same way.
(iii)	Minimal surprise	Users should not be surprised by the behaviour of a system.
(iv)	User guidance	The interface should provide meaningful feedback when errors take place and offer context-sensitive user help facilities.
(v)	Recoverability	The interface should have mechanisms to permit users to recover from errors.
(vi)	User diversity	The interface should provide suitable interaction facilities for various kinds of system user.

Table 3.2 illustrates the design principles on the basis of the human capabilities. These general principles are applicable to all user interface designs and should normally be instantiated as more detailed design directions for specific organizations.

(i) User Familiarity – The principle of user familiarity says that users should not be forced to change an interface. The interface should use user familiar terms and system manipulated objects should directly be related to the user's environment.

(ii) Consistency – It suggests that system commands and menus should have the identical format; parameters should be passed to all commands in the similar manner; and the punctuation of commands should be same. It helps user in learning fast. Also, knowledge learnt in one command or application can be used in other parts.

(iii) Minimal Surprise – It is much appropriate principle because users are annoyed when a system works in an unexpected way. During the use, users build a mental model of system's working. If a command in one context causes a particular type of change, it is quite obvious to expect the same command in different context to cause a comparable change. But, if something completely different takes place, the user is surprised and confused as well. Therefore, interface designers must ensure that comparable actions have comparable effects.

(iv) Recoverability – This is an important principle because users usually make mistakes while using a system. These mistakes can be minimized by the interface design but cannot be completely eliminated. User interfaces should facilitate users to recover from their mistakes. They can provide the following two types of facilities –

(a) Confirmation of Destructive Actions – If users specify a potentially destructive action, they should be asked that " is really what they want". This avoids information to be destroyed.

(b) The Provision of an Undo Facility – Before occurring the action, the "Undo" restores the system to a state. Since, users don't always identify immediately that a mistake has been made, multiple levels of undo are useful.

(v) User Guidance – There should be built-in user assistance or help facilities in interfaces. These should be integrated with the system and should provide several different levels of assistance and suggestion, ranging from basic information on getting started to a full description of system facilities. They should be structured and there should not be any overwhelming information, when users ask for assistance.

(vi) User Diversity – User diversity principle identifies that, for many interactive systems, there may be various kinds of users. Some of them are casual users who occasionally interact with the system, while others may be power users who make use of the system daily for several hours. The former category of users requires interfaces providing guidance while the latter needs shortcuts permitting them to interact as quickly as possible. Moreover, there may be different types of disability that users may face and (if possible) the interface should be adaptable to cope up with these.

The requirement of the user-interface design depends on the type of end-users. There are mainly three types of end-users –

(i) Novice User – Novice is having little or no knowledge about the physical computer system or about computer usage in general. Vocabulary restriction to a small number of familiar, consistently used terms is essential for their user. Initial focus on a few simple tasks to build confidence and reduce anxiety. Feedback that is clear and informative is essential. Carefully designed user manuals and on-line help are also effective.

(ii) Knowledgeable Intermittent User – Generally understands the task and computer concepts well but often gets hung up with the execution part because of the intermittent usage issues. Using a simple and consistent structure focusing on recognition rather than recall is important. Consistent sequence of actions, meaningful messages and frequent prompts encourage comfort.

(iii) Frequent User – Frequent users seek to get their work done rapidly. These users demand rapid response times, brief unobtrusive feedback and shortcuts.

Q.32. Explain user interface design process. ***(R.G.P.V., Dec. 2010)***

Ans. The user interface design process is iterative in nature and can be represented by spiral model as shown in fig. 3.9.

As shown in fig. 3.9, the user interface design process consists of following four different activities –

(i) User, task, and environment analysis and modeling

(ii) Interface design

(iii) Interface construction

(iv) Interface validation.

The spiral in fig. 3.9 indicates that each of the above noted tasks will take place more than once, with each pass around the spiral representing extra enlargement of users requirements and the resultant design.

The initial analysis activity emphasizes on the user's profile. Skill level, business understanding, and general receptiveness to the new systems are recorded while different categories of users are defined, for which requirements are elicited. In short, the software engineer tries to understand the system perception for each category of users.

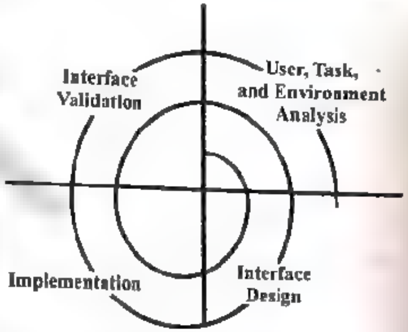


Fig. 3.9 The User Interface Design Process

Once defining general requirements, a more detailed task analysis is performed. These tasks, for the accomplishment of the goals of the system, are identified, described, and elaborated.

The user environment analysis concentrates on the physical work environment. Following are some questions to be asked during analysis –

(i) Where will be interface be located physically ?

(ii) Does the interface hardware accommodate space, light, or noise constraints ?

(iii) Will the user be sitting, standing, or performing other tasks unrelated to the interface ?

(iv) Are there special human factors considerations driven by environmental factors ?

The information collected during analysis activity is used to create an analysis model for the interface. This model is used as a basis to begin design activity.

The purpose of interface design is to determine a set of interface objects and commands through which a user can perform all the defined tasks in such a way that satisfies every usability goal defined for the system.

The implementation activity usually commences with the prototype creation enabling usage scenarios to be evaluated. With the iterative design process, a user interface tool kit may be used to complete the interface construction.

Validation emphasizes on the following aspects –

(i) The ability of the interface to correctly implement every task of the user, to accommodate all variations in the task, and to get all general user requirements.

(ii) The users acceptance of the interface as a useful tool in their work.

(iii) The degree to which the interface is easy to use and easy to learn.

Q.33. Discuss the core activities involved in user interface design process. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.8 and Q.32.

Q.34. Explain the process of user interface design. Also provide guidelines for good user interface. ***(R.G.P.V., June 2017)***

Ans. Refer to Q.32 and Q.31.

Q.35. Explain user interface design and UML. ***(R.G.P.V., June 2016)***

***Ans.* User Interface Design** – Refer to Q.29 and Q.32.

UML – Refer to Q.18.

Q.36. Discuss the major design issues of a user interface design.

Ans. Following are the four common design issues of a user interface design –

(i) System response time (ii) User help facilities

(iii) Error information handling (iv) Command labeling.

The above noted design issues must be established and considered at the beginning of software design, when changes are easy and costs are low.

These are described below –

(i) System Response Time – The response time is measured from the point where some control action begins, until getting desired output from the software. It has two characteristics –

(a) Length – The lengthy response time results in user frustration and stress. While a short response time can also be detrimental in case of user being paced by the interface.

(b) Variability – Low deviation from average response time helps user to have an interaction rhythm, even for long response time.

(ii) User Help Facilities – Two different types of help facilities are as follows –

(a) Integrated Help Facility – It is context sensitive and enables the user to choose from the topics relevant to the currently performed actions. This reduces user time required to obtain help and increases the friendliness of the interface.

(b) Add-on Help Facility – It occurs after completion of the software. It is an on-line user's manual, having limited query capability.

(iii) Error Information Handling – Every error message or warning delivered by an interactive system should have the following characteristics –

(a) The message should be such that the user can understand the problem.

(b) The message should give constructive advice to cope up with the problem.

(c) The message should indicate the negative consequences of the error. This helps user to check and ensure that they have not taken place.

(d) The message should have an audible or visual cue. It may be a beep generated to accompany the display of the message, or the message might flash momentarily, or some other way that helps recognizing the error.

(e) The message should be nonjudgemental i.e., the working should never place blame on the user.

(iv) Command Labeling – In the past times, the typed command was the most common type of interaction between user and the system software. Thus, it was commonly used for every application. The following design issues arise when using typed commands –

(a) What form will commands take ?

(b) Will every menu option have a corresponding command ?

(c) Can commands be customized or abbreviated by the user ?

(d) How difficult will it be to learn and remember the commands ? What can be done, in case of forgetting a command ?

Q.37. Write a short note on function-oriented design. ***(R.G.P.V., June 2014)***

Ans. Function-oriented design is the result of focusing attention to the function of the program. In function-oriented design, the design consists of module definitions, with each module supporting a functional abstraction. In a function-oriented design approach, a system is viewed as a transformation function, transforming the inputs to the desired outputs. The purpose of the design phase is to specify the components for this transformation function, so that each component is also a transformation function. Hence, the basic output of the system design phase, when a function-oriented design approach is being followed, is the definition of all the major data structures in the system, all the major modules of the system, and how the modules interact with each other.

Q.38. What is cohesive module ? ***(R.G.P.V., June 2016)***

Ans. A cohesive module performs only one task in software procedure with little interaction with other modules. In other words, cohesive module performs only one thing. A strongly cohesive module implements functionality that is related to one feature of the solution and needs little or no interaction with other modules. Cohension may be viewed as glue that keeps the module together. It is a measure of the mutual efficity of the components of a module. Thus, we want to maximize the interaction within a module. Hence, an important design objective is to maximize the module cohesion and minimize the module coupling.

Q.39. Differentiate between abstraction and modularization. ***(R.G.P.V., June 200***

Ans. Abstraction of a component is a view of the component that extracts the essential information relevant to the particular purpose and ignores the rest of the information. It allows a module to be specified by its external behaviour only. Abstraction permits a designer to concentrate on one component at a time by using the abstraction of other components to decide the interaction between components.

Modularity is a means of problem partitioning in software design. A system is considered modular if each component has a well-defined abstraction and a change in one component has minimal impact on other components.

Q.40. What do you mean by stepwise refinement ? (R.G.P.V., June 200

Or

What do you understand by the term top down decomposition in the context of function-oriented design ? Explain your answer with suitable example. ***(R.G.P.V., June 201***

Ans. Stepwise refinement is a top-down design method. It was originally proposed by Niklaus Wirth. The development of a program is done by successively refining levels of procedural detail. The development of a hierarchy is done by decomposing a macroscopic statement of function in a stepwise manner until programming language statements are reached.

The program refinement process proposed by Wirth is similar to the process of refinement and partitioning that is used during requirements analysis. The distinction is in the level of implementation detail that is taken into account, not the approach.

Actually a process of elaboration is the refinement. We start with a statement of function that is defined at a high level of abstraction. It means that, the statement describes function or information conceptually but gives no information about the internal workings of the function or the internal structure of the information. Refinement causes the designer to elaborate on the original statement, offering more and more detail since each successive refinement takes place.

The concepts of abstraction and refinement are complementary. Both concepts helps the designer in creating a complete design model as the design evolves.

For example, suppose a function ***create_new_library member*** which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This high-level function may be refined into the following subfunctions –

(i) assign_membership_number

(ii) create_member_record

(iii) print_bill.

Each of these sub-functions, may be split into more detailed subfunctions, and so on.

Q.41. Write short note on information hiding. (R.G.P.V., Dec. 2006)

Or

Discuss the relationship between the concept of information hiding as an attribute of effective modularity and the concept of module independence. ***(R.G.P.V., Dec. 2008)***

Ans. The ***information hiding*** principle states that modules can be characterized by design decisions that (each) hides from all others. That is, modules should be specified and designed in such a way that information held within a module is not accessible to other modules having no need for such information.

Hiding specifies that effective modularity can be obtained by defining a set of independent modules that communicate with each other only when information necessary to get software function. Abstraction aids to define the procedural entities that build up the software. Hiding defines and enforces access constraints to both procedural detail within a module and any local data structure used by the module.

Using information hiding as a design criterion for modular systems offers the greatest advantages when modifications are needed during testing and later during maintenance. Inadvertent errors are less likely to propagate to other software locations because most of the information (data and procedure) are hidden from the other software parts.

Q.42. Explain the term functional independence. (R.G.P.V., Dec. 2003)

Ans. The ***functional independence*** is a direct outgrowth of modularity and the abstraction and information hiding concepts. It can be achieved by creating modules with single-minded function. In other words, a developer wants to design software in a way that each module addresses a specific subfunction of requirements and having a simple interface. Independence is quite important in software designing, since a software having independent modules is easier to develop, as function may be compartmentalized and interfaces are simplified. In essence, functional independence is a key to effecti design, and design is the key to software quality.

Two qualitative criteria are used to measure independence – cohesion an coupling. Coupling is a measure of the relative interdependence among modul Cohesion is a measure of the relative functional strength of a module.

Q.43. Explain the term cohesion. ***(R.G.P.V., Dec. 200***

Or

Write short note on cohesion. ***(R.G.P.V., June 200***

Ans. Cohesion is a measure of the strength of the relationships betwe responsibilities of the components of a module. A module is said to be high cohesive if its components are strongly related to each other by some mea of communication or resource sharing or the nature of responsibilities. Henc the cohesion may be understood as a measure of relatedness of elements of module, as shown in fig. 3.10. High cohesion is a symbol of good design as facilitates execution of a task by maximum intra-modular communication an least inter-module dependencies. It increases interdependencies betwee different modules and hence it contrasts with coupling.

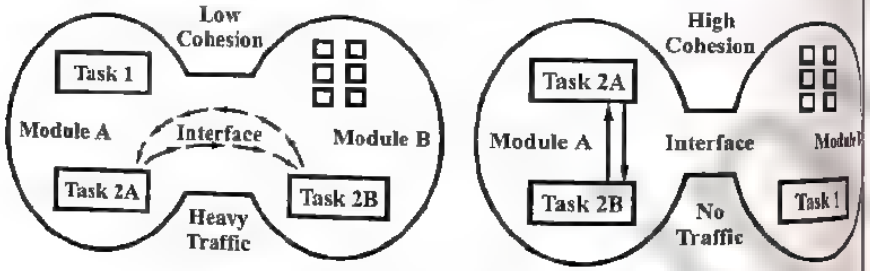


Fig. 3.10 Low and High Cohesion

Designers should create strong, highly cohesive modules, i.e. the modul whose elements are strongly related to each other. On the other hand, th elements of one module should not be strongly related to the elements o another module, because that would lead to tight coupling between the modul

Q.44. Briefly explain each level of cohesion. How can you determi the cohesion level of a module ?

Ans. Different levels of cohesion are described as follows –

(i) Co-incidental – Co-incidental cohesion occurs if there is n meaningful relationship among the module elements. It can also occur in th case of modularizing an existing program by chopping it in several pieces an making different pieces modules. If a module is made to save duplicate cod by combining some segment of code occurring at several places, that modu is very much likely to have co-incidental cohesion. In such a case, the modu

statements have no relationship among them. Moreover, if one of the modules using the code is to be modified and this modification includes the common code, it may happen that other modules using the code do not need to modify it. Hence, this "common module" modification may result in incorrect behaviour of other modules. Thus, these modules have strong interconnections and are not easily modifiable.

(ii) Logical – A module has logical cohesion, if it is having some logical relationship between its elements and the elements perform functions of the same logical class. A module performing all the inputs or all the outputs is a very good example of logical cohesion. With such a module, if we want to input or output a particular record, we have to convey this to the module. It can be done by passing some kind of special status flag, used to determine the statements to be executed in the module. Generally, logically cohesive modules are avoided, if possible.

(iii) Temporal – It is similar to the above explained logical cohesion, except that the elements are also related in time and are simultaneously executed. Modules performing "initialization", "clean-up", and "termination" activities are usually temporally bound. Although, the elements in a temporally bound module are logically related, temporal cohesion is higher than logical cohesion. It is because, the elements of logically bound module are all executed together, and thus avoids the problem of passing the flag making the code simpler.

(iv) Procedural – A procedurally cohesive module is having elements belonging to a common procedural unit. For example, a loop or a sequence of decision statements in a module may be combined to get an individual (separate) module. Procedurally cohesive modules, in general, occur after determining the modular structure from a flowchart. Procedural cohesion often cuts across functional lines.

(v) Communicational – Communicationally cohesive modules have elements related by a reference to the same input or output data i.e., in such modules, the elements are together because they operate on the same input or output data. For example a module to "print and punch record". Communicationally cohesive modules may perform several functions, but it is sufficiently high as to be generally acceptable.

(vi) Sequential – In this cohesion, the elements are together in a module because the output of one provides the input to another. But, the sequential cohesion does not provide any guidelines to combine the elements into modules, if we have a sequence of elements with output of one is treated as input of the other. Sequentially cohesive modules bear a close resemblance to the problem structure. However they are considered to be far from ideal.

(vii) Functional – It is the strongest cohesion. In a functionally cohesive module, all the module elements are related to perform a single function, including a single goal accomplishment. Some examples are, "compute square root" and "sort the array".

Now the cohesion level of a module can only be found by using our judgement, there is no mathematical formula for this. A useful method for deciding whether a module has functional cohesion is to write sentence that describes, completely clearly, the function of the module. Then the following tests can be made –

(a) If the sentence must be a compound sentence, (if it is having a comma, or it has more than one verb), the module is probably performing several functions, and having a sequential or communicational cohesion.

(b) If the sentence is having time related words, such as "first", "next", "when", and "after" the module probably has sequential or temporal cohesion.

(c) If the predicate of the sentence does not have a single specific object following the verb, the module probably has logical cohesion.

(d) If it is having words like "initialize" and "cleanup", temporal cohesion may be the case.

Q.45. What is meant by cohesion ? How should a software be designed considering cohesion ? ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.43 and Q.44.

Q.46. What is coupling ? ***(R.G.P.V., June 2016)***

Ans. It is the strength of interconnections between modules. It is a measure of interdependence among modules. Generally the more we must know about module X so as to understand module Y, the more firmly bounded X is to Y. Highly coupled modules are joined by strong interconnections, while loosely coupled modules have weak interconnections, and independent modules have no interconnections. For separately solving and modifying the modules, we want them to be loosely coupled. This selection of modules decides the coupling between them. Coupling between modules is largely decided during system design rather than implementation, since the system modules are created during this time.

In designing software, we try hard for lowest possible coupling. Simple connectivity among modules provides software that is easier to understand and less prone to a ***ripple effect***, caused when errors takes place at one location and propagate through a system.

Coupling is an abstract concept and is hardly quantifiable. Thus, there are no formulas to determine the coupling between modules. However, some major factors can be identified as influencing coupling between modules, These are –

(i) The type of connection between modules

(ii) The complexity of the interface

(iii) The type of information flow between modules.

With the increase in complexity and obscurity of the module interfaces, coupling increases. Thus, minimizing the number of interfaces per module and their complexity, coupling can be kept low.

Another factor affecting coupling is the interface complexity. The more is the interface complexity, the higher will be the degree of coupling. Some predetermined level of interface complexity is, however, required for the modules communication. But, somehow, more than this minimum is used. Essentially, we should keep the interface as simple and small as possible.

The third major factor affecting coupling is the type of information flow along the interfaces. These are of two kinds i.e., data and control. Sending or receiving control information implies that the action of the module will depend on it, while transferring data information indicates that a module passes some data (as input) to another module and accepts in return some data as output. Thus, a module can be treated as a simple input-output function, performing some transformations on input data to get the output data.

Q.47. What do you mean by terms cohesion and coupling in the context of software design ? How are these concepts useful in arriving at a good design of a system ? ***(R.G.P.V., June 2014)***

Or

Differentiate between cohesion and coupling with example. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.43 and Q.46.

Q.48. Enumerate the different types of coupling that might exist between the two modules. Give examples of each. ***(R.G.P.V., June 2013)***

Ans. Examples of different types of coupling are shown in fig. 3.11. Modules x and p are subordinate modules. Each is unrelated and hence no ***direct coupling*** takes place. Module z is subordinate to module x and is accessed through a conventional argument list. through which data is passed. The low coupling called, ***data coupling*** is shown in this portion of structure as long as simple argument list is present. When a portion of a data structure is passed through a module interface (between x and y), a variation of data coupling called ***stamp coupling***, is found. At moderate level. ***control coupling*** is common, where a control flag is passed between modules p and q. When modules are tied to an environment external to software relatively, high level of coupling takes place. ***External coupling*** is necessary, but should be constrained to few modules with a structure. When a number of modules reference a global data area high coupling also takes place ; ***common coupling***, at this mode is called, is depicted in fig. 3.11. Modules z, r, and v each access a data item in a global data area. ***Content coupling*** is the highest degree of coupling. It takes place when one module makes use of data or control information maintained within the boundary of another module. Secondarily content coupling takes place when branches are made into the module middle. This coupling should be avoided.

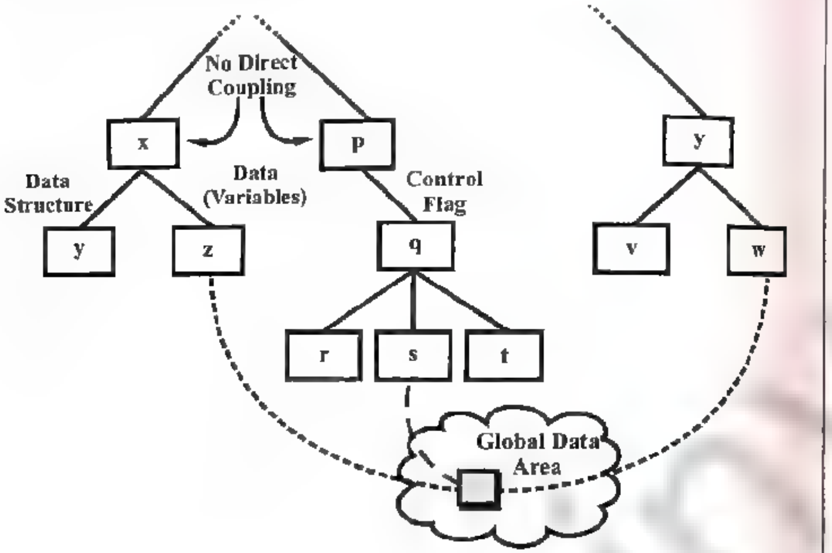


Fig. 3.11 Types of Coupling

Q.49. What is functional independence ? Explain different cohesion and coupling types in detail. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.42, Q.44 and Q.48.

Q.50. Define the module coupling and explain different type of coupling. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.46 and Q.48.

Q.51. What is modularity ? List out the important properties of modular system in brief. ***(R.G.P.V., June 2015, Dec. 2015)***

Ans. The real power of a design process comes from modularity i.e., if a system is partitioned into modules, it is easily solvable and modifiable. Moreover, it will be much better if the modules are also separately compilable. A system is said to be modular, if it is having discrete components such that each component can be separately implemented, and there is minimum impact of change to one component over other components.

Modularity is thus a desirable property in a system. It helps in system debugging, system repair, and in system building.

A software system cannot be made modular by simply chopping it into several modules. For modularity, each of the several modules needs to support a well-defined abstraction. Also, each module should have a clear interface through which it can interact with others.

Five criteria have been suggested by Meyer that enable us to evaluate a design method with respect to its ability to define an effective modular system.

Modular Decomposability – If a systematic mechanism is provided by a design method to decompose the problem into subproblems, it will reduce the complexity of the overall problem, hence achieving an effective modular solution.

Modular Understandability – If a module is considered as a standalone unit i.e., without reference to other modules, it will be easy to build and easy to change.

Modular Composability – If existing (reusable) design components can be assembled by a design method into a new system, it will provide a modular solution that does not reinvent the wheel.

Modular Continuity – The smaller changes in the system requirements result in changes in the individual modules, instead of system wide changes, the impact of change-induced side effects will be minimized.

Modular Protection – If an unacceptable condition takes place within a module and its effects are limited within that module, the impact of error-induced side effects will be minimized.

Q.52. Explain the concept of modularity in design engineering process. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.13 (iii) and Q.51.

Q.53. Explain the term modularity. What is cohesion and coupling ? Explain with examples. ***(R.G.P.V., June 2010, 2011)***

Ans. **Modularity** – Refer to Q.51.

Cohesion and Its Types – Refer to Q.43 and Q.44.

Coupling and Its Types – Refer to Q.46 and Q.48.

Q.54. What is modularity ? List important properties of modular design. What problems are likely to arise if two modules have

(i) High coupling (ii) Low cohesion. ***(R.G.P.V., June 2008)***

Ans. **Modularity and Properties of Modular Design** – Refer to Q.51.

Problems – A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program. Stated simply, a cohesive module should (ideally) do just one thing.

As an example of low cohesion, consider a module that performs error processing for an engineering analysis package. The module is called when computed data exceeds prespecified bounds. It performs the following tasks-

(i) Computes supplementary data based on original computed data,

(ii) Produces an error report (with graphical content) on the user's workstation,

(iii) Performs follow-up calculations requested by the user,

(iv) Updates a database,

(v) Enables menu selection for subsequent processing.

Although the preceding tasks are loosely related, each is an independent functional entity that might best be performed as a separate module. Combining the functions into a single module can only serve to increase the likelihood of error propagation when a modification is made to one of the processing tasks noted above.

Coupling refers to a measure of interconnection among modules in any software. Coupling relies on the interface complexity between modules, the point at which entry or reference is made to a module and what data pass across the interface. Complicated and dense (high) connectivity among modules results in high coupling, which in turn results in software that is difficult to understand and more prone to a "ripple effect" that causes the errors that take place at one location to propagate throughout the system, resulting in a chaos. In addition, highly coupled modules are difficult to understand and trouble shoot (debug).

Q.55. Write short notes on –

(i) Modularization

(ii) Coupling, cohesion

(iii) Design verification.

(R.G.P.V., Dec. 2016)

Ans. (i) Modularization – Refer to Q.51.

(ii) Coupling, Cohesion – Refer to Q.46 and Q.43.

(iii) Design Verification – The output of the system design phase should be verified before proceeding with the activities of the next phase. Unless the design is specified in a formal, executable language, the design cannot be executed for verification. Other means for verification have to be used. The most common approach for verification is design reviews or inspections.

Design reviews ensure that the design satisfies the requirements and is of good quality. If the errors are made during the design process, they will ultimately reflect themselves in the code and the final system. It is best if design errors are detected early before they manifest themselves in the system as the cost of removing faults caused by errors that occur during design increases with the delay in detecting the errors. In design review process a group of people get together to discuss the design with the aim of revealing design errors or undesirable properties. The meeting ends with a list of action items, which are later acted on by the design team. Further reviews may be organized, if needed.

Q.56. What are the various strategies of design ? Which is most popular and practical ? ***(R.G.P.V., June 2005)***

Or

Differentiate between top-down and bottom-up design.

(R.G.P.V., June 2008)

Or

Explain the types of software design strategies available.

(R.G.P.V., Dec. 2016)

Ans. A system is actually consisting of components, having components of their own. In other words, a system is a hierarchy of components. The highest-level component corresponds to the total system. Such a hierarchy can be designed by two possible approaches i.e., ***top-down*** and ***bottom-up***. The former one starts from the highest-level and moves towards the lower level of the hierarchy, while in contrast, the latter starts with the lowest-level component and proceeds through progressively higher-levels to the top-level component of the hierarchy.

A top-down approach starts by identifying and decomposing the major system components into their lower-level components. This process repeats until the desired level of detail is achieved. Top-down design strategies often result in some form of stepwise refinement, which is the process in which starting from an abstract design, in each step the design is refined to a better level, until reaching the level where no more refinement is needed and the design can directly be implemented. The top-down approach has been found to be extremely useful for design. Most of the methodologies are based on this approach.

A bottom-up approach of design starts with designing the most primitive components and proceeds to higher-level components using lower-level components. This method works with layers of abstraction. Beginning from the very bottom, operations providing a layer of abstraction are implemented. The operations of this layer are then implemented in more powerful operations and a still higher layer of abstraction, until the stage comes where the operations supported by the layer are the desired ones.

A top-down approach is appropriate in case where the specifications of the system are clearly known. However, if a system is to be built from an existing system, a bottom-up approach is more appropriate.

Pure top-down or pure bottom-up approach is often not practical. For being successful using a bottom-up approach, we must have a good notion of the top towards which design should be proceeding. Unless having a good idea about the needed operations at the higher layers, it would be difficult to determine what operations the current layer should support. On the other hand, top-down approaches require some idea concerning the feasibility of the components specified during design. Also, these components should be implementable which needs some idea concerning the feasibility of the lower-level parts of a component. An approach combining the two (i.e., top-down and bottom-up) approaches is providing a layer of abstraction for the application domain via., libraries of function having the functions of interest to the application domain.

Q.57. If some existing modules are to be re-used in building a new system, which design strategy is used and why ? ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.56.

Q.58. Discuss various approaches of partitioning.

Or

Discuss how structural partitioning can help to make software more maintainable. ***(R.G.P.V., June 2005)***

Ans. If the system is having hierarchical architecture, the program structure can easily be partitioned both horizontally and vertically. Fig. 3.12 represents this view.

In fig. 3.12 (a), horizontal partitioning defines individual branches of the modular hierarchy for every major program function. Control modules (shown by rectangles) are used to coordinate communication between and execution of the functions. Three partitions are made in simple horizontal partitioning i.e., input, data transformation (processing), and output. The following benefits are provided by horizontal partitioning –

(i) Software, that can easily be tested

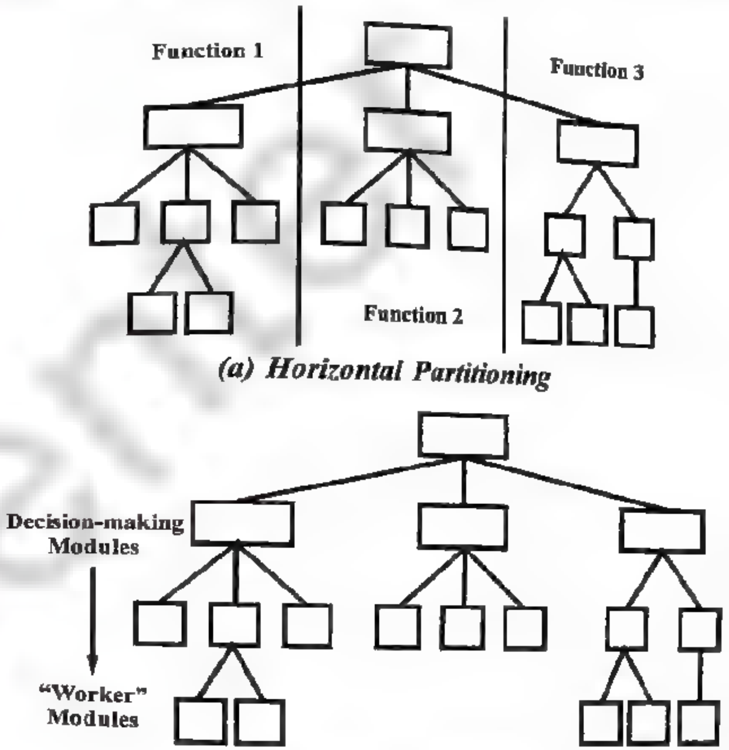
(ii) Software, that can easily be maintained

(iii) Propagation of fewer side effects

(iv) Software, that can easily be extended.

On the other hand, vertical partitioning, also called "factoring", says that control and work should be distributed in the program structure, top to down-

Top-level modules have to perform control functions and do less processing work, while lower modules in the structure are the workers, performing all input, processing, and output tasks.



(a) Horizontal Partitioning

(b) Vertical Partitioning

Fig. 3.12 Structural Partitioning

Fig. 3.12 (b) represents vertical partitioning. The need for vertical partitioning is due to the nature of change in program structures. Observing the fig. 3.12 (b), it can be said that a change in control module will have a greater chance of propagating side effects to its subordinates. While, a change to a worker module (lower level) is having less chance to propagate side effects. Generally, changes are – changes to input, computation or transformation, and output. The basic behaviour of the program is much less likely to change. That is why, vertically partitioned structures are less susceptible to side effects due to changes and thus be more maintainable, which is its key quality factor.

Q.59. Discuss function-oriented design techniques. ***(R.G.P.V., June 2005, Dec. 2014)***

Or

Differentiate between structured analysis and structured design in the context of function-oriented design. ***(R.G.P.V., June 2011, 2013)***

Ans. The design activity begins when the requirements document for the software to be developed is available and the architecture has been designed. In a function-oriented design approach, a system is viewed as a transformation function, transforming the inputs to the desired outputs. The purpose of the design phase is to specify the components for this transformation function, so that each component is also a transformation function That is, each module in design supports a functional abstraction. The basic output of the system design phase, when a function-oriented design approach is being followed, is the definition of all the major data structure in the system, all the major modules of the system, and how the modules interact with each other.

Some of the function-oriented software design techniques are –

(i) Structured Analysis – During structured analysis, the major processing tasks (functions) of the system are analyzed, and all the data flowing among these processing tasks are represented graphically. This methodology is based on the following underlying principles –

(a) Top-down decomposition approach.

(b) Divide and conquer principle, where each function is isolated from other functions and then decomposed individually into subfunctions totaly disregarding the effects on other functions.

(c) Graphical representation of the analysis results.

Structured analysis encompasses the following activities –

(a) The SRS document is examined to determine –

(1) Different high-level functions that the system needs to perform.

(2) Data input to every high-level function.

(3) Data output from every high-level function.

(4) Interactions (data flow) among the identified.

These aspects of the high-level functions are then represented in a diagramatic form, This forms the top-level Data Flow Diagram (DFD), usually called the DFD 0.

(b) Each high-level function is decomposed into its constituent subfunctions through the following set of activities –

(1) Different subfunctions of the high-level function are identified.

(2) Data input to each of these subfunctions are identified.

(3) Data output from each of these subfunctions are identified.

(4) Interactions (data flow) among these subfunctions are identified.

These aspects are then represented in a diagramatic form using a DFD.

(c) Step (b) is repeated recursively for each subfunction until a subfunction can be represented by using a simple algorithm.

The results of step (a) and each iteration through step (b) of structured analysis are usually represented using a DFD. A DFD in simple words, is a hierarchical graphical model of a system that depicts the different processing activities or functions of the system and the data interchange among these processes.

(ii) Structured Design – The aim of structured design is to transform the results of structured analysis (i.e., a DFD representation) into a structure chart. A structure chart represents the software architecture i.e., the various modules making up the system, the module dependency (i.e., which module calls which other modules), and the parameters that are passed among the different modules. Thus, the structure chart representation can be easily implemented using some programming language. Since the main focus in a structure chart representation is on the module structure of a software and the interaction among the different modules, the procedural aspects (e.g., how a particular functionality is achieved) are not represented.

Q.60. Give a brief overview of SA/SD methodology. Distinguish between structured analysis and structured design in context of function-oriented design. ***(R.G.P.V., June 2010)***

Ans. SA/SD methodology involves carrying out two distinct activities – structured analysis (SA) and structured design (SD). Fig. 3.13 shows the roles of structured analysis (SA) and structured design (SD). In this figure, the structured analysis activity transforms the SRS document into a graphic model known as the DFD model. During structured analysis, functional decomposition of the system is done. It means that, each function that the system require to perform is analyzed and hierarchically decomposed into more detailed functions. In contrast, during structured design, all functions recognized during structured analysis are mapped to a module structure. This module structure is known as the software architecture or the high-level design for the given problem. This is represented with the help of a structure chart.

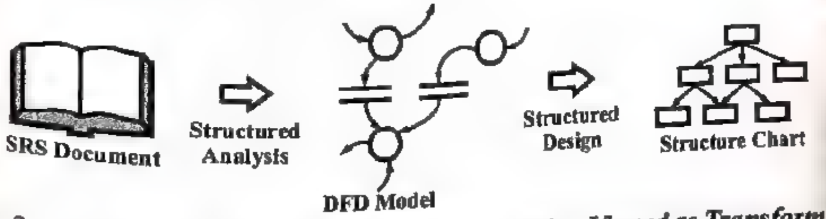


Fig. 3.13 Structured Analysis and Structured Design Viewed as Transformers

Structured Analysis and Structured Design – Refer to Q.59.

Q.61. Explain about component based design. (R.G.P.V., Dec. 2014)

Ans. Component based development or component based software engineering emerged in the late 1990s as a reuse-based approach to software systems development. Its motivation was frustration that object-oriented development has not led to extensive reuse as originally suggested. Single object classes were too detailed and specific and had to be bound with an application either at compile-time or when the system was linked. Detailed knowledge of the classes were required to use them and this usually meant that source code had to be available. This presented difficult problems for marketing components. In spite of early optimistic predictions, no significant market for individual objects has ever developed.

Components are more abstract than object classes and can be considered to be stand-alone service providers. When a system needs some service, it calls on a component to provide that service without caring about where that component is executing or the programming language used to develop the component. For example, a very simple component could be a single mathematical function that computes the square root of a number When a program requires a square root computation, it calls on that component to provide it. At the other end of the scale, a system that needs to carry out some arithmetic computations could call on a spreadsheet component that provides a calculation service.

Q.62. Discuss in detail data dictionary. (R.G.P.V, June 2005, 2009)

Ans. Data objects, function, and control are the three representations that the analysis model encompasses. In each representation, data objects and control items play a role (or, either one of them). Thus, it is essential to provide such an organizational approach that represents the characteristics of each data object and control item. This is the task accomplished by "data dictionary".

The data dictionary is a quasi-formal grammar that describes the content of objects defined during structured analysis. This important data modeling notation is defined more precisely in the following way –

"The ***data dictionary*** is an organized listing of all data elements that are appropriate to the system, with precise, accurate definitions so that both the analyst and the customer will have a common understanding of inputs, outputs, components of stores, and even intermediate calculations."

In the present scenario, the data dictionary is always implemented as a part of a CASE structured analysis and design tool. Although having difference in format from tool to tool, most of the data dictionary formats contain the following information –

(i) ***Name*** – Provides the primary name of the data or control item, the data store, or an external entity.

(ii) ***Alias*** – These are the other names used in place of the previous one, for the first entry.

(iii) ***Where-used/How-used*** – It is a listing of the processes that use data objects and entities and how they are used in the system. For example, input to the process, output from the process, as a store, as an external entity, etc.

(iv) ***Content Description*** – It is a notation that represents content of objects.

(v) ***Supplementary Information*** – There are other information about data types, preset values, restrictions or limitations, and so on.

Consistency in naming can be enforced after entering the name and aliases of a data object or control item in a data dictionary. It means that if any team member of analysis decides to name a newly generated data item "AB", but "AB" already exists in the dictionary, then the CASE tool (supporting the dictionary) sends a warning to indicate duplicate names. This enhances the analysis model consistency and thus helps to eliminate errors.

"Where-used/how-used" is the information recorded automatically from the flow models. When a dictionary entry is made, the CASE tool checks DFDs and CFDs to know which processes use the data/control information and also it determines how it is used. Although appearing unimportant, it is actually a very important benefit of the data dictionary.

The notation for getting content description is given in the following table –

Table 3.3 Content Description Table

Data Construct	*Notation*	*Meaning*
	=	is composed of
Sequence	+	and
Selection	[|]	either-or
Repetition	{ }^{n}	n repetitions of
	( )	optional data
	...	delimits comments

The content describing notation enables a software engineer to show composite data in one of the following three fundamental ways –

(i) As a selection from among a data items set.

(ii) As a sequence of data items.

(iii) As a repeated grouping of data items. Each data item entry representing a part of a sequence, selection, or repetition may itself be another composite data item that requires further refinement within the dictionary.

Q.63. What are specific tools for structured analysis ? Write the demerits of system flowchart. ***(R.G.P.V., June 2006)***

Ans. The specific tools for structured analysis are –

(i) E-R Diagram

(ii) DFD

(iii) Control flow model

(iv) Control specification

(v) Process specification

(vi) Decision table

(vii) Data dictionary.

Demerits of System Flowchart –

(i) It is difficult to identify the different modules of the software from its flowchart representation.

(ii) Data interchange among different modules is not represented in a flowchart.

(iii) System flowchart does not provide exact information about the data types and memory requirements of the variables used in the modules.

(iv) System flowcharts are very time-consuming and laborious to draw with proper symbols and spacing, especially for large complex programs.

(v) Owing to the symbol-string nature of flowcharting, any changes or modifications in the program logic will usually require a completely new flowchart. Redrawing a flowchart is again so tedious that many companies either do not redo them or produce the flowchart by using a computer program to draw it. There are several computer programs available that will read the program's instructions and draw a flowchart of its logic, but these programs are fairly expensive to acquire and use a lot of computer time.

(vi) There are no standards determining the amount of detail that should be included in a flowchart.

Because of such limitations, many organizations are now reducing the amount of flowcharting used.

Q.64. Explain structured design methodology for developing system design. ***(R.G.P.V., Dec. 2010)***

Ans. The major concern of the design phase is creating the software system design. Over the years, many design techniques have been proposed to provide some descipline in handling the complexity of designing large systems. The main aim of design methodologies is not to reduce the process design to a sequence of mechanical steps but to provide the guidelines during the design process to aid the designer.

The structure design methodology (SDM) views every software system as having some inputs which are converted into the desired output by the software system. The software is viewed as a transformation function which transforms the given inputs into the desired outputs and the central problem of designing software systems to be properly designing this transformation function. The structured design methodology is primarily function oriented, due to this view of software and it relies heavily on functional abstraction and functional decomposition.

The concept of the structure of a program lies at the heart of the structured design method. The structure design methodology aims to control and influence the structure of the final program, during design. The main aim is to design a system such that programs implementing the design would have a fine hierarchical structure, with the functionally cohesive modules and as some interconnections between modules are as possible.

Actually, in properly designed systems, the case in which a module with subordinates does not perform much computation. Its subordinates performed the bulk of actual computation and the module itself largely coordinates the data flow between the subordinates to get the computation done. The subordinates can get the bulk of their work done by their subordinates until the"atomic" modules, that have no subordinates are reached. The process of decomposing a module is ***factoring*** that the bulk of its work is done by its subordinates. If all the actual processing is accomplished by bottom-level atomic modules and also if the nonatomic modules largely perform the jobs of control and coordination, then a system is said to be completely factored. To achieve a structure which is close to being completely factored is attempted by structured design methodology.

The complete strategy is to identify the input and output streams and the primary transformations which have to be performed to produce the output. Then high level modules are created to perform these major activities.

In this strategy, there are four major steps as follows –

(i) Restate the problem as a data flow diagram

(ii) Identify the input and output data elements

(iii) First-level factoring

(iv) Factoring of input, output and transform branches.

A part from the function oriented design methodology, object oriented (OO) approaches for software development have become extremely popular in recent years. OO system offers many advantages. An OO model closely represents the problem domain, which makes it easier to produce and understand designs. As requirements change, the objects in a system are less immune to these changes, thereby permitting changes more easily. Inheritance and close association of objects in design to problem domain entities encourage more reuse, that is, new applications can use existing modules more effectively, thereby reducing development cost and cycly time. Object oriented approaches are believed to be more natural and provide richer structures for thinking and abstraction. Common design patterns have also been uncovered that allow reusability at a higher level.

The object-oriented design approach is fundamentally different from the function-oriented design approaches primarily due to the different abstraction that is used. It requires a different way of thinking and partitioning. It can be said that thinking in object-oriented terms is most important for producing truly object-oriented designs.

Q.65. Write a short note on design metrics. ***(R.G.P.V., June 2014)***

Ans. The success of a software project depends largely on the quality and effectiveness of software design. Hence, it is important to develop software metrics from which meaningful indicators can be derived. With the help of these indicators, necessary steps are taken to design software according to user requirements. Various design metrics, like architectural design metrics, metrics for object-oriented design, component-level design metrics, and user-interface design metrics are used to indicate the complexity, quality, and so on of software design.

Q.66. Describe the metrics for the design model of a product. What are the attributes of effective software metrics ? ***(R.G.P.V., June 2017)***

Ans. (i) Design Metrics – Refer to Q.65.

(ii) Architectural Design Metrics – These metrics focus on features of program architecture which stress on architectural structure and effectiveness of components within the architecture. These metrics do not require any inner working knowledge of any software module.

In architectural design metrics, three software design complexity measures are defined, these are –

(a) In hierarchical structure, structural complexity for module 'j' can be calculated as –

$$S(j) = f_{out}^2(j)$$

where, $f_{out}(j)$ = Fan-out of module 'j'.

(b) Data complexity is complexity in internal interface for module 'j' can be calculated as –

$$D(j) = v(j)/[f_{out}(j) + 1]$$

where, v(j) = Number of input and output variable in 'j'.

(c) System complexity is sum of structural and data complexity

i.e., $$C(j) = S(j) + D(j)$$

(iii) Component-level Design Metrics – These metrics focuses on software component's internal characteristics. The software engineer can judge the quality of component-level design by measuring "three Cs"

(a) Cohesion Metrics – Cohesiveness of module can be indicated by definition of following five concepts –

(1) Data Slice – It is defined as a backward walk through a module, which looks for value of data that affect the state of module as the walk start.

(2) Data Tokens – It is defined as a set of variables defined for a module.

(3) Glue Tokens – It is defined as a set of data tokens, which lies on one or more data slice.

(4) Superglue Tokens – It is defined as tokens which are present in every data slice in module.

(5) Stickiness – It is defined as stickiness of glue token, which depends on number of data slices that it binds.

(b) Coupling Metrics – These metrics indicates the degree of which the module is connected to other modules, global data and other outside environments.

(c) Complexity Metrics – Different types of software metrics can be calculated to ascertain the complexity of program control flow. Cyclomatic complexity is one of the most widely used complexity metrics for ascertaining the complexity of the program.

(iv) Use Interface Design Metrics – Layout appropriateness and cohesion metric are commonly used user interface design metrics. Layout appropriateness provides absolute and relative positions of all layout entities like icons, text, menus, windows and so on. It also provide frequency with which layout entity is used and cost of change over from one layout entity to another. Cohesion metric for user interface measures on-screen content to another on-screen content.

(v) ***Object-oriented Design Metrics*** – The characteristics of object-oriented design metrics are -

(a) Size – It is defined by four views such as population, volume, length and functionality.

(b) Complexity – It is determined by assessing how classes are related to each other.

(c) Coupling – It is a physical connection between object-oriented design elements.

(d) Sufficiency – The degree to which an abstraction possesses the features required of it.

(e) Cohesion – It is determined by analyzing the degree to which a set of properties that the class possesses is part of the problem domain or design domain.

(f) Primitiveness – It is a degree to which the operation is atomic.

(g) Similarity – It indicate similarity between classes in terms of thier structure, function, behaviour or purpose.

(h) Volatility – Probability of occurrence of change in the object-oriented design.

Attributes of Effective Software Metrics – Following are the attributes of effective software metrics –

(i) Simple and computable

(ii) Objective and consistent

(iii) Consistent in use of units and dimensions

(iv) Programming language independent

(v) High quality

(vi) Easy to calibrate

(vii) Robust – Insensitive to minor product change

(viii) Value – Should be available at reasonable cost

(ix) Validation – Should measure what it is intended to measure.

Q.67. Discuss briefly the network metrics.

Ans. Network metrics for design focus on the structure chart and define some metrics of how "good" the structure or network is in an effort to quantify the complexity of the call graph. As coupling of a module increases if it is called by more modules, a good structure is considered one that has exactly one caller. That is, the call graph structure is simplest if it is a pure tree. The more the structure chart deviates from a tree, the more complex the system. Deviation of the tree is then defined as the ***graph impurity*** of the design. Graph impurity can be defined as

Graph impurity = $n - e \quad 1$

where n is the number of nodes in the structure chart and e is the number of edges. As in a pure tree the total number of nodes is one more than the number of edges, the graph impurity for a tree is 0. Each time a module has a fan-in of more than one, the graph impurity increases. This is the major drawback of this approach. It ignores the common use of some routines like library or support routines. In the design evaluation tool that we use, we do not consider the lowest-level nodes for graph impurity because we believe that most often the lowest-level modules are the ones that are used by many different modules, particularly if the structure chart was factored. Library routines are also at the lowest level of the structure chart.

Q.68. Do you agree with the following assertion ? "A design solution that is difficult to understand would lead to increased development and maintenance cost". Give reasonings for your answer. (R.G.P.V., June 2014)

Ans. A good design should help overcome the human cognitive limitations the arise due to limited short-term memory. A large problem overwhelms the human mind, and a poor design would make the matter worse. Unless a design solution is easily understandable, it could lead to an implementation having a large number of defects and at the same time tremendously pushing up the development costs. Therefore, a good design solution should be simple and easily understandable. A design that is easy to understand is also easy to develop and maintain. A complex design would lead to severely increased life cycle costs. Unless a design is easily understandable, it would require tremendous effort to implement test, debug and maintain it. About 60% of the total effort in the life cycle of a typical product is sepent on maintenance. If the software is not easy to understand not only would it lead to increased development costs, the effort required to maintain the product would lead to a program that is full of bugs and is unreliable. The understandability of a design solution can be enhanced through clever applications of the principles of abstraction and decomposition.

Q.69. Describe various metrics that can be used to evaluate function oriented design. (R.G.P.V., June 2010)

Ans. The general process metrics of interest from each major step are the cost or total effort spent in the activity, the schedule, and the number and distribution of errors detected by the reviews. The cost and schedule metrics are needed for tracking if the project is progressing according to the plan, and if not, what actions should be performed by the project management. These are tracked throughout the phase, though at the end of a defined phase, its effect on the overall project schedule and cost can be better understood. If the cost and schedule of the phase is different from what was specified in the plan, this implies that either the project is not being managed properly and hence their are cost and schedule overruns or that the earlier estimates were incorrect. In the first case, closer monitoring and tighter control results. In the second case, cost and schedule are estimated again after the design, and if the revised estimates of cost and schedule are different from earlier estimates, a renegotiation of the project and modification of the plan results.

Quality is monitored generally through reviews. If the process is under statistical control, one can expect density and distribution of errors found in reviews in the design of this project to be similar to what was found in earlier projects. Using the part history about design reviews and the current review data, the quality of the design can be estimated. At least, it can be ensured that the design of this product is as good as the design of earlier projects done using the existing process.

Size is always a product metric of interest, as size is the single most influential factor deciding the cost of the project. As the actual size of the project is known only when the project ends, at early stages the project size is only an estimate. Our ability to estimate size becomes more accurate as development proceeds. Hence, after design, size (and cost) reestimation is done by project management. After design, as all the modules in the system and major data structures are known, the size of the final system can be estimated quite accurately.

For estimating the size, the total number of modules is an important metric. This can be obtained from the design. By using an average size of a module from this metric the final size in LOC can be estimated. Alternatively, the size of each module can be estimated, and then the total size of the system will be estimated as the sum of all the estimates. As a module is a small, clearly specified programming unit, estimating the size of a module is relatively easy.

Besides the size metric, the other product metric of interest at design time is design quality. The simplicity is the most important design quality attribute as it directly affects the testability and maintainability of the software. Complexity metrics are used to quantify simplicity.

Q.70. Discuss stability metrics to quantify the complexity of design.

Ans. Stability of a design is a metric that tries to quantify the resistance of a design to the potential ripple effects that are caused by changes in modules. The higher the stability of a program design, the better the maintainability of the program.

At the lowest level, stability is defined for a module. From this, the stability of the whole system design can be obtained. The aim is to define a measure so that the higher the measure the less the ripple effect on other modules that in some way are related to this module. The modules that can be affected by change in a module are the modules that invoke the module or share global data (or files) with the module. Any other module will clearly not be affected by change in a module. The potential ripple effect is defined as the total number of assumptions made by other modules regarding the module being changed. Hence, counting the number of assumptions made by other modules is central to determining the stability of a module.

As at design time only the interfaces of modules are known and not their internals, for calculating design stability only the assumptions made about the interfaces need be considered. The interface of a module consists of all elements through which this module can be affected by other modules, i.e., through which this module can be coupled with other modules. Hence, it consists of the parameters of the modules and the global data the module uses. Once the interface is identified, the structure of each element of the interface is examined to determine all the ***minimal*** entities in this element for which assumptions can be made. The minimal entities generally are the constituents of the interface element. For example, a record is broken into its respective fields as a calling module can make assumptions about a particular field.

For each minimal entity at least two categories of assumptions can be made – about the type of the entity and about the value of the entity. Each minimal entity in the interface is considered as contributing one assumption in each category. A structured type is considered as contributing one more assumption about its structure in addition to the assumptions its minimal elements contribute. The procedure for determining the stability of a module x and the stability of the program can be broken into a series of steps.

Step 1 – From the design, analyze the module x and all the modules that call x or share some file or data structure with x, and obtain the following sets.

J_x = {modules that invoke x}

J'_x = {modules invoked by x}

R_{xy} = {passed parameters returned from x to y, $y \in J_x$}

R'_{xy} = {parameters passed from x to y, $y \in J'_x$}

GR_x = {Global data referenced in x}

GD_x = {Global data defined in x}

Note that determining GR_x and GD_x is not always possible when pointers and indirect referencing are used. In that case, a conservative estimate is to be used. From these, for each global data item i, define the set G_i as

$$G_i = \{x | i \in GR_x \cup GD_x\}$$

The set G_i represents the set of modules where the global data i is either referenced or defined. Where it is not possible to compute G accurately, the worst case should be taken.

Step 2 – For each module x, determines the number of assumptions made by a caller module y about elements in R_{xy} (parameters returned from module x to y) through these steps –

(a) Initialize assumption count to 0.

(b) If i is a structured data element, decompose it into base types, and increment the assumption count by 1; else consider i minimal.

(c) Decompose base types, and if they are structured, increment the count by 1.

(d) For each minimal entity i, if module y makes some assumption about the value of i, increment the count by 2; else increment by 1.

Let TP_{xy} represent the total number of assumptions made by a module about parameters in R_{xy}.

Step 3 – Determine TP'_{xy}, the total number of assumptions made by module y called by the module x about elements in R'_{xy} (parameters passed from module x to y). The method for computation is the same as in the previous step.

Step 4 – For each data element $i \in GD_x$ (i.e., the global data elements modified by the module x), determine the total number of assumptions made by other modules about i. These will be the modules other than x that use or modify i, i.e., the set of modules to be considered is $\{G_i - \{x\}\}$. The counting method of step 2 is used. Let TG_x be the total number of assumptions made by other modules about the elements in GD_x.

Step 5 – For a module x, the design logical ripple effect (DLRE) is defined as –

$$DLRE_x = TG_x + \sum_{y \in J_x} TP_{xy} + \sum_{y \in J_x} TP'_{xy}.$$

$DLRE_x$ is the total number of assumptions made by other modules that interact with x through either parameters or global data. The design stability (DS) of a module x is then defined as

$$DS_x = 1/(1 + DLRE_x).$$

Step 6 – The program design stability (PDS) is computed as

$$PDS = 1 \Big/ \left(1 + \sum_{x} DLRE_x\right)$$

By following this sequence of steps, the design stability of each module and the overall program can be computed. The stability metric, in a sense, is trying to capture the notion of coupling of a module with other modules. The stability metrics can be used to compare alternative designs – the larger the stability the more maintainable the program. It can also be used to identify modules that are not very stable and that are highly coupled with other modules with a potential of high ripple effect. Changes to these modules will not be easy, hence a redesign can be considered to enhance the stability.

Q.71. Discuss the information flow metrics to quantify the complexity of design.

Ans. The information flow metrics attempt to define the complexity in terms of the total information flowing through a module, in an effort to quantify coupling.

The earliest work on information flow metrics was done by Henry and Kafura. In their metric, the complexity of a module is considered as depending on the intramodule complexity and the intermodule complexity. The intramodule complexity is approximated by the size of the module in lines of code. The intermodule complexity of a module depends on the total information flowing in the module (*inflow*) and the total information flowing out of the module (*outflow*). The inflow of a module is the total number of abstract data elements flowing in the module (i.e., whose values are used by the module), and the outflow is the total number of abstract data elements that are flowing out of the module (i.e., whose values are defined by this module and used by other modules). The module design complexity, D_c, is defined as –

$$D_c = size * (inflow * outflow)^2.$$

The term (inflow*outflow) refers to the total number of combinations of input source and output destination. This term is squared, as the interconnection between the modules is considered a more important factor determining the complexity of a module. This is based on the common experience that the modules with more interconnections are harder to test or modify compared to other similar-size modules with fewer interconnections.

The metric defined earlier defines the complexity of a module purely in terms of the total amount of data flowing in and out of the module and the module size. A variant of this was proposed based on the hypothesis that the module complexity depends not only on the information flowing in and out but also on the number of modules to or from which it is flowing. The module size is considered an insignificant factor, and complexity D_c for a module is defined as –

$$D_c = fan\_in*fan\_out + inflow*outflow$$

where fan_in represents the number of modules that call this module and fan_out is the number of modules this module calls.

UNIT 4 SOFTWARE ANALYSIS AND TESTING

SOFTWARE STATIC AND DYNAMIC ANALYSIS, CODE INSPECTIONS

Q.1. What is static analysis of software ?

Ans. Static analysis comprises a set of methods used for analyzing the software source code or object code to understand how the software functions and to set up criteria to check its correctness. Static analysis studies the source code without executing it and reveals a variety of information like the structure of model used, data and control flows, syntactical accuracy, and much more.

Q.2. Discuss the types of static analysis methods.

Ans. There are several types of static analysis methods –

(i) Control Analysis – This focuses on examining the controls used in software calling structure, control flow analysis, and state transition analysis. The calling structure deals with the model by identifying the calling and the called structure. The calling structure can be procedure, subroutine, function, or method. The control flow analysis checks the sequence of control transfers. In addition, it identifies incorrect and inefficient constructs in the model. A graph of the model is constructed in which the conditional branches and model junctions are represented by nodes.

(ii) Data Analysis – This ensures that proper operations are applied to data objects such as data structures and linked lists. In addition, this method also ensures that the defined data is properly used. Data analysis comprises two methods, namely, data dependency and data-flow analysis. Data dependency is essential for assessing the accuracy of synchronization across multiple processors. Data flow analysis checks the definition and references of variables.

(iii) Fault/Failure Analysis – This analyzes the fault (incorrect model component) and failure (incorrect behaviour of a model component) in the model. This method uses input-output transformation descriptions to identify the conditions that are cause for the failure. To determine the failures in certain conditions the model design specification is checked.

(iv) Interface Analysis – This verifies and validates the interacting and distributive simulations to check the software code. There are two basic techniques for the interface analysis, namely, model interface analysis and user interface analysis. Model interface analysis examines the submodel interfaces and determines the accuracy of the interface structure. User interface analysis examines the user interface model and checks for precautionary steps taken to prevent errors during the user's interaction with the model. The method also concentrates on how accurately the interface is integrated into the overall model and simulation.

Q.3. What is dynamic analysis of software ?

Ans. For dynamic testing the test objects are executed or simulated. Dynamic analysis is what is generally considered as testing i.e., it involves running the system.

Dynamic testing is an imperative process in the software life cycle. Every procedure, every module and class, every subsystem, and the overall system must be tested dynamically, even if static tests and program verifications have been carried out.

The activities for dynamic testing are –

(i) Preparation of the test object for error localization

(ii) Availability of a test environment

(iii) Selection of appropriate test cases and data

(iv) Test execution and evaluation.

Q.4. Differentiate static and dynamic testing done. State few salient characteristics of modern testing tools. ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.1 and Q.3.

Some characteristics of modern testing tools are as follows –

(i) It should support GUI based test preparation.

(ii) These tools should be able to adopt the underlying hardware.

(iii) It should be easy to use.

(iv) It should use one or more testing strategy for performing testing on host as well as on target platform.

(v) It should provide complete code coverage and create test documentation in various formats (HTML/DOC/RTF .....).

(vi) It should provide a clear report on test case, steps, test case status (PASS/FAIL).

Q.5. Distinguish between the static and dynamic analysis of a program. Explain at least one metric test a static analysis tool reports and at least one metric that a dynamic analysis reports. How are these metrics useful ? ***(R.G.P.V., June 2013)***

Ans. **Differences between the Static and Dynamic Analysis** – Refer to Q.1 and Q.3.

Static analysis tools often summarize the results of analysis of every function in a polar chart known as Kiviat chart. A Kiviat chart typically represents the analyzed values for cyclomatic complexity, number of source lines, percentage of comment lines, Halstead's metrics, etc.

In general, the dynamic analysis results are reported in the form of a histogram or pie chart to describe the structural coverage achieved for different modules of the program. The output of a dynamic analysis tool can be stored and printed easily to offer evidence that thorough testing has been carried out.

Q.6. What is meant by code walkthrough ? What are some of the important types of errors checked during code walkthrough ? ***(R.G.P.V., June 2013)***

Ans. Code walkthrough is an informal code analysis technique. In this technique, a module is taken up for review after the module has been coded, successfully compiled, and all syntax errors have been eliminated. A few members of the development team are given the code a couple of days before the walkthrough meeting. Each member selects some test cases and simulates execution of the code by hand. The members note down their findings of their walkthrough and discuss those in a walkthrough meeting where the coder of the module is present.

Even though code walkthrough is an informal analysis technique, several guidelines have evolved over the years for making this naive but useful analysis technique more effective. These guidelines are based on personal experience, common sense, several other subjective factors. Therefore, these guidelines should be considered as examples rather than as accepted rules to be applied dogmatically. Some of these guidelines are the following –

(i) The team performing code walkthrough should not be either too big or too small. Ideally, it should consist of between three to seven members.

(ii) Discussions should focus on discovery of errors and avoid deliberations on how to fix the discovered errors.

(iii) In order to foster cooperation and to avoid the feeling among the engineers that they are being evaluated in the code walkthrough meetings, managers should not attend the walkthrough meeting.

The main objective of code walkthrough is to discover the algorithmic and logical errors in the code.

Q.7. Explain the code inspection. ***(R.G.P.V., June 201***

Or

What is code inspections ? ***(R.G.P.V., June 201***

Ans. Code inspection technique is a formal and systematic examination of the source code to detect errors. During this process, the software is presented to the project managers and the users for a comment of approval. Before providing any comment, the inspection team checks the source code for errors. Generally, this team consists of a moderator, reader, recorder, and author.

(i) Moderator – It conducts inspection meetings, checks errors and ensures that the inspection process is followed.

(ii) Reader – The reader takes the team through the code by paraphrasing its operation. Never let the author take this role, since he may read what he meant instead of what he implemented.

(iii) Recorder – It keeps record of each error in the software code. This frees the task of other team members to think deeply about the software code.

(iv) Author – It observes the code inspection process silently and helps only when explicitly required. The role of the author is to understand the errors found in the software code.

Generally, to conduct code inspections, a number of steps are followed such as –

(i) Planning – After the code is compiled and there are no more errors and warning messages in the software code, the author submits the findings to the moderator who is responsible for forming the inspection team. After the inspection team is formed, the moderator distributes the listings as well as other related documents like design documentation to each team member. The moderator plans the inspection meetings and coordinates with the team members.

(ii) Overview – This is an optional step and is required only when the inspection team members are not aware of the functioning of the project. To familiarize the team members, the author provides details to make them understand the code.

(iii) Preparation – Each inspection team member individually examines the code and its related materials. They use a checklist to ensure that each problem area is checked. Each inspection team member keeps a copy of this checklist, in which all the problematic areas are mentioned.

(iv) Inspection Meeting – This is carried out with all team members to review the software code. The moderator discusses the code under review with the inspection team members.

(v) Rework – The author makes all suggested corrections, gets a clean compile and sends it back to the moderator.

(vi) Follow-up – The moderator checks the reworked code. If the moderator is satisfied the inspection is formally complete and the code may be tested.

SOFTWARE TESTING FUNDAMENTALS, SOFTWARE TEST PROCESS, TESTING LEVELS, TEST CRITERIA, TEST CASE DESIGN, TEST ORACLES, TEST TECHNIQUES, BLACK-BOX TESTING, WHITE-BOX UNIT TESTING AND UNIT TESTING FRAMEWORKS, INTEGRATION TESTING, SYSTEM TESTING AND OTHER SPECIALIZED TESTING, TEST PLAN, TEST METRICS, TESTING TOOLS

Q.8. What is testing ? Why is it important ? Explain error, fault and failure. ***(R.G.P.V., June 2010)***

Ans. Software testing is a process of ensuring acceptable degree of quality attributes of software. It's purpose is to ensure correctness, robustness, reliability, and many other quality attributes. Testing is the process of finding the errors introduced at any stage of development. It may also be defined as the process of executing the software under a stipulated environment as well as out of bounds with an intention of finding the errors in it. Although the testing can show the presence of an error, it cannot certify the absence of an error.

The IEEE defines the software testing as the process of analysing a software item to detect differences between existing and required conditions and to evaluate the features of the software item.

Reviews and other SQA activities can and do uncover errors, but they are not adequate. When the program is executed, the customer tests it. Therefore, the program is executed before it gets to the customer with the desire of finding and eliminating all errors. In order to get the maximum possible number of errors, test must be carried out systematically and test cases must be designed through disciplined techniques.

Error is used in two different manners. Error refers to the discrepancy between a computed, observed, or measured value and the true, specified, or theoretically correct value. That is, error refers to the difference between the actual output of a software and the correct output. In this interpretation, error is essentially a measure of the difference between the actual and the ideal. Error is also used to refer to human action that results in software containing a defect or fault. This definition is quite general and encompasses all the phases. When the system gives the result in an unexpected manner then the system error occurs.

Fault is a situation that causes a system to fail in performing its required function. A fault is the basic reason for software malfunction and is synonymous with the commonly used term bug. The term error is also often used to refer to defects. Fault is the probability of the system that the failure can lead to system error.

Failure is the inability of a system or component to perform a required function according to its specifications. A software failure occurs if the behaviour of the software is different from the specified behaviour. Failures may be caused due to functional or performance reasons. A failure is produced only when there is a fault in the system. However, presence of a fault does not guarantee failure. In other words, faults have the potential to cause failures and their presence is a necessary but not a sufficient condition for failure to occur. It may also be possible that a failure may occur but not be detected. When the system does not perform as per the user expectations, then system failure occurs.

Q.9. What are the error, fault and failure regarding to system ? And why it occurs in system ? ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.8.

Q.10. Distinguish between an error and a failure in the context of program testing. Justify your answer. ***(R.G.P.V., June 2013)***

Ans. Refer to Q.8.

Q.11. What are the primary objectives of software testing ?

Ans. The primary objectives of software testing are as follows –

(i) Testing is a process of executing a program to find an error and

(ii) A good test case should have a high probability of finding an as-yet-undiscovered error.

(iii) A test case will be considered successful if it uncovers an as-yet-undiscovered error.

The objective of a software engineer is to design tests that systematically uncover different categories of errors and to accomplish this taking minimum amount of time and effort.

If successfully conducted, testing will uncover errors in the software. Moreover, it demonstrates that software functions seem to be working accordingly, and that the behavioural and performance requirements seem to have been fulfilled. Furthermore, data collected during testing provide an indication of software reliability and quality.

Q.12. What are software testing principles ? ***(R.G.P.V., June 2009)***

Ans. The basic principles of software testing are as follows –

(i) *Testing should depend on user requirement.* This is in order to uncover any defects that might cause the program or system to fail to meet the client's requirements.

(ii) *It is impossible to test everything.* Exhaustive tests of all possible scenarios are impossible because of the many different variables affecting the system and the number of paths a program flow might take.

(iii) *Test planning should be done early.* This is because test planning can start independently of coding and as soon as the client requirements are set.

(iv) *Test for invalid and unexpected input conditions as well as valid conditions.* The program should generate correct messages when an invalid test is encountered and should generate correct results when the test is valid.

(v) The probability of the existence of more errors in a module or group of modules is directly proportional to the number of errors already found.

(vi) *Testing time and resources are limited.* Avoid redundant tests.

(vii) Testing should begin at the module. The focus of testing should be concentrated on the smallest programming units first and then expand to other parts of the system.

(viii) Testing must be done by an independent party. Testing should not be performed by the person or team that developed the software since they tend to defend the correctness of the program.

(ix) Assign best personnel to the task, because testing requires high creativity and responsibility only the best personnel must be assigned to design, implement and analyze test cases, test data, and test results.

(x) Testing should not be planned under the implicit assumption that no error will be found.Testing is the process of executing software with the intent of finding errors.

Q.13. The software analysis and design are constructive task and software testing is considered to be destructive from the developer point of view. Discuss. ***(R.G.P.V., June 2017)***

Ans. From the developer point of view, software analysis and design are constructive tasks because a software developer creates a software program with its data structures and documentation. The software developer is proud of the software that has been built. Just like any builder who built an edifice. When testing start, if anything found complex, difficult to understand, yet definite then there will be attempt of breaking things that the builder or software developer has built. So that from the developer point of view, testing is considered to be destructive.

Q.14. Explain software testing process and different steps involved in it.

Ans. A testing process is usually performed in cycles. The first cycle begins with some obvious tests related to the functionalities of the system as specified in its specifications. Later more formal tests are planned for the detailed testing of the system. While designing test cases for detailed testing, the primary basis is the attribute to be tested. Test cases are planned and executed for all important functionalities and attributes of the system. Results are captured and summarized reports documented. In the next cycle, the first cycle reports are referred for reviewing the system and its response for test cases. Debuggers provide a report for all fixed and not-fixed bugs. The comments for not-fixed bugs are reviewed and may be resuggested in the next report. Testing of the updated system is performed based on the last reports and additional test cases. In further cycles, more formal testing is executed in a more aggressive and formal manner with manual or automated or both means. The testing process goes stricter and stricter in the subsequent cycles to expose more hidden errors.

A formal testing process covers activities, such as review of program plans, development of the formal test plan, creation of test documentation (test design, test cases, test software, and test procedures), and acquisition of automated testing tools, test execution, updating the test documentation, and tailoring the model for projects of all sizes (see fig. 4.1).

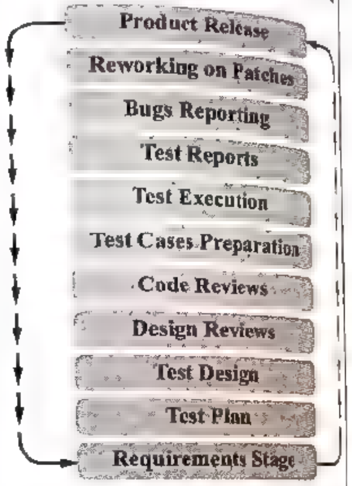


Fig. 4.1 Software Testing Life Cycle

Microsoft developes network suggests the following steps while testing a system –

(i) Prepare comprehensive test plan specifications and test cases for each level of testing. Supplement these with the test data and test logs. Test plans for system testing may involve operators and test plans because acceptance testing involves customers.

(ii) Design the test cases to test system restrictions, such as file and database size.

(iii) Develop the data to test specific cases. Copies of live files must not be used except for in acceptance testing.

(iv) Do not use confidential data for testing without written authorization, especially in the case of acceptance testing.

(v) Follow relevant standards.

(vi) Perform regression testing on each component of the system. This ensures that no anomalies have crept into the system because of the changes made to the system.

(vii) Make sure to document and set up the test environment for each level in advance of testing. Test environments specify the preconditions required to perform the tests.

(viii) Specify the intended test coverage as part of the test plan. Test coverage is the degree to which specific test cases address all specified requirements for a specific system or component.

Q.15. What are the fundamentals of software testing and discuss the characteristics of a good test ? ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.14 and Q.4.

Q.16. Write short note on levels of testing. ***(R.G.P.V., June 2004)***

Or

What are the different levels of testing and goals at the different levels ? For each level specify which of the testing approaches is most suitable. ***(R.G.P.V., June 2010)***

Ans. Different levels of testing are used in the testing process. Each level of testing aims to test different aspects of the system.

The basic levels are unit testing, integration testing, and system and acceptance testing. These different levels of testing attempt to detect different types of faults. Fig. 4.2 shows the relation of the faults introduced in different phases, and the different levels of testing.

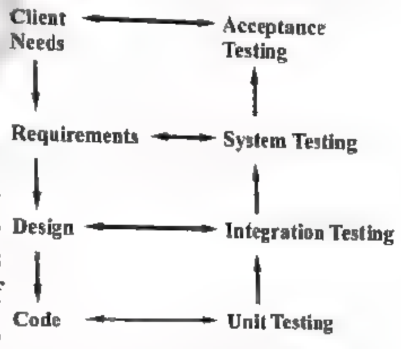


Fig. 4.2 Levels of Testing

The first level of testing is called ***unit testing.*** In this, different modules are tested against the specifications produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules. It is typically done by the programmer of the module. A module is considered for integration and use by others only after it has been unit tested satisfactorily.

Due to its close association with coding, the coding phase is frequently called "coding and unit testing". As the focus of this testing level is on testing the code, structural testing is best suited for this level. In fact, as structural testing is not very suitable for large programs, it is used mostly at the unit testing level.

The next level of testing is often called ***integration testing.*** In this, many unit tested modules are combined into subsystems, which are then tested. The goal here is to see if the modules can be integrated properly. Thus, the emphasis is on testing interfaces between modules.

The next levels are ***system testing*** and ***acceptance testing.*** Here the entire software system is tested. The reference document for this process is the requirements document, and the goal is to see if the software meets its requirements. Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily. Here testing focuses on the external behaviour of the system; the internal logic of the program is not emphasized. Consequently, more functional testing is performed at these levels.

These levels of testing are performed when a system is being built from the components that have been coded. There is another level of testing, called ***regression testing***, that is performed when some changes are made to an existing system. However, as modifications have been made to an existing system, testing also has to done to make sure that the modification has not had any undesired side effect of making some of the earlier services faulty. That is, besides ensuring the desired behaviour of the new services, testing has to ensure that the desired behaviour of the old services is maintained. This is the task of regression testing.

Q.17. Explain test criteria. ***(R.G.P.V., June 2010)***

Ans. Test selection criterion or simply test criterion is used for –

(i) How should we select our test cases ?

(ii) On what basis should we include some element of the program domain in the set of test cases and not include others ?

For a given program P and its specifications S, a test selection criterion specifies the conditions that must be satisfied by a set of test cases T. The criterion becomes a basis for test case selection.

There are two aspects of test case selection – specifying a criterion for evaluating a set of test cases and generating a set of test cases that satisfy a given criterion. Many test case criteria have been proposed. However, generating test cases for most of these is not easy and cannot, in general, be automated fully. Often a criterion is specified and the tester has to generate test cases that satisfy the criterion. In some cases, guidelines are available for deciding test cases. Overall the problem of test case selection is very challenging and current solutions are limited in scope.

There are two fundamental properties for a testing criterion – *reliability* and *validity*. A criterion is reliable if all the sets (of test cases) that satisfy the criterion detect the same errors. That is, it is insignificant which of the sets satisfying the criterion is chosen; every set will detect exactly the same errors. A criterion is valid if for any error in the program there is some set satisfying the criterion that will reveal the error. A fundamental theorem of testing is that if a testing criterion is valid and reliable, if a set satisfying the criterion succeeds (revealing no faults) then the program contains no errors. However, it has been shown that no algorithm exists that will determine a valid criterion for an arbitrary program.

Q.18. What are test cases ? ***(R.G.P.V., Dec. 2010)***

Ans. A test case is a document that describes an input, action, or event and its expected result, in order to determine whether the software or a part of software is working correctly or not.

IEEE defines test case as 'a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement'. Generally, a test case contains particulars, such as test case identifier, test case name, its objective, test conditions/setup, input data requirements, steps, and expected results.

Incomplete and incorrect test cases lead to incorrect and erroneous test outputs. To avoid this, a test case should be developed in such a manner that it checks a software with all possible inputs. This process is known as ***exhaustive testing*** and the test case, which is able to perform exhaustive testing, is known as ***ideal test case.***

Generally, a test case is unable to perform exhaustive testing therefore, a test case that gives satisfactory results is selected. In order to select a test case, certain questions should be addressed.

(i) How to select a test case ?

(ii) On what basis are certain elements of program included or excluded from a test case ?

Q.19. Explain test case design. ***(R.G.P.V., May 2018)***

Ans. Test cases in software testing will integrate the set of the input to the system, its conditions of execution, and its expected output, which will be compared with the captured actual output after execution of the system in the described conditions and with the input mentioned in the test case. Format of reporting a test case and its fail/pass criteria is very important. Different formats are used by different organizations. These formats may also change depending on the type of project. Fig. 4.3 shows a general format.

Test ID	Test Description	Input Description	Execution Condition	Expected Output	Captured Output	Fail/Pass Result	Remarks

Fig. 4.3 General Test Case Format

The test ID will identify a test case in the test document and test reports. The test description covers the common description of the local feature or functionality for the test case and the steps to be taken for running the test case and capturing the output. Execution conditions define the preconditions of the system for the test case, assumptions for state of the system for the test case, and environment constraints to be applied for the system to test with the test case. Expected output describes the expected behaviour of the system and its expected results of processing. This may also include describing the pass/fail criteria of the test case. Captured output will have the output and behaviour of the system as captured while its actual execution for the test case under its predefined conditions and input of the test case fail/pass result will show whether the test case failed or passed as per its fail/pass criteria. Remarks will contain the review comments of the tester and any other detail of the system for the test case.

Q.20. Explain the test oracles. *(R.G.P.V., June 2010, 2011)*

Ans. A test oracle is a mechanism, different from the program itself, that can be used to check the correctness of the output of the program for the test cases. Conceptually, we can consider testing a process in which the test cases are given to the test oracle and the program under testing. The output of the two is then compared to determine if the program behaved correctly for the test cases. This is shown in fig. 4.4.

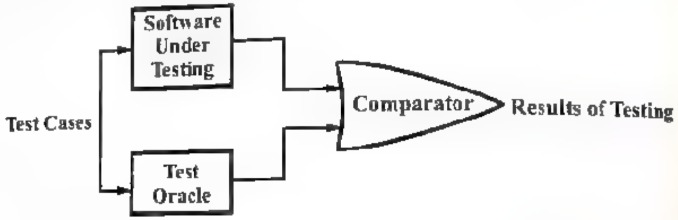


Fig. 4.4 Testing and Test Oracles

Test oracles are necessary for testing. Ideally, we would like an automated oracle, which always gives a correct answer. However, often the oracles are human beings, who mostly compute by hand what the output of the program should be. As it is often extremely difficult to determine whether the behaviour conforms to the expected behaviour, our "human oracle" may make mistakes. As a result, when there is a discrepancy between the results of the program and the oracle, we have to verify the result produced by the oracle, before declaring that there is a fault in the program.

The human oracles generally use the specifications of the program to decide what the correct behaviour of the program should be. To help the oracle determine the correct behaviour, it is important that the behaviour of the system or component be unambiguously specified and that the specification itself is error free. In other words, the specification should actually specify the true and correct system behaviour.

There are some systems where oracles are automatically generated from specifications of programs or modules. With such oracles, we are assured that the output of the oracle is consistent with the specifications. However, even this approach does not solve all our problems, because of the possibility of errors in the specifications. Consequently, an oracle generated from the specifications will only produce correct results if the specifications are correct, and it will not be dependable in the case of specification errors. Furthermore, such systems that generate oracles from specifications are likely to require formal specifications, which are frequently not generated during design.

Q.21. What is overall strategy for software testing ? Explain it clearly. *(R.G.P.V., June 2017)*

Ans. Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software technique – a set of steps into which we can place specific test case design techniques and testing methods – should be defined for the software process.

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system function against customer requirements. A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems must surface as early as possible.

A strategy for software testing may also be viewed in the context of spiral as shown in fig. 4.5.

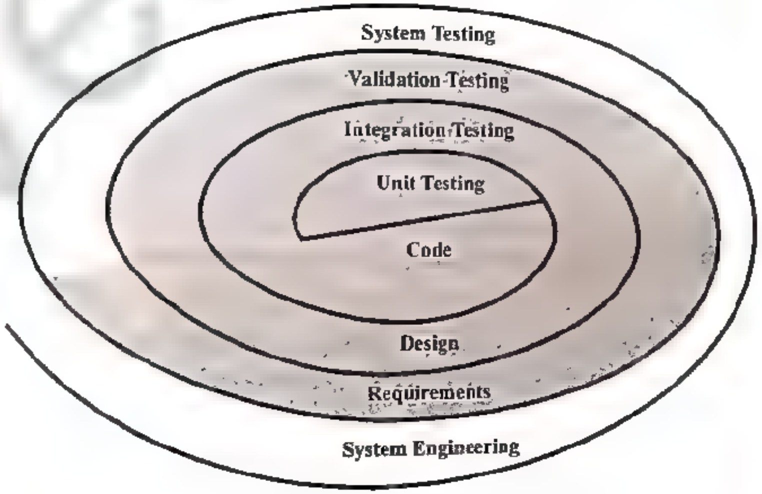


Fig. 4.5 Software Testing Strategy

Unit testing begins at the vortex of spiral and concentrates on each unit of the software as implemented in source code. Testing progresses by moving outward along the spiral to ***integration testing*** where the focus is on design and the construction of the software architecture. Then we encounter ***validation testing***, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at ***system testing***, where the software and other system elements are tested as a whole. To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.

Q.22. Discuss software testing strategies. Differentiate between verification and validation. ***(R.G.P.V., Dec. 2014)***

Ans. **Software Testing Strategies** – Refer to Q.21.

Difference between Verification and Validation – The terms verification and validation are often used interchangeably, but they have different meanings verification is the process of determining whether or not the products of a given phase of software development fulfill the specification established during the previous phase. Verification activities include proving, testing, and reviews. Validation is the process of evaluating software at the end of the software development to ensure compliance with the software requirements. Boehm states this another way

Verification – "Are we building the right product ?"

Validation – "Are we building the product right ?".

Q.23. How software inspection improves software quality ? Explain software inspection process in brief. ***(R.G.P.V., May 2018)***

Ans. The term software inspection was developed by IBM in the early 1970s, when it was noticed that the testing was not enough sufficient to attain high quality software for large applications.

Inspection is used to determine the defects in the code and remove it efficiently. It prevents the defect and raise the quality of testing to remove the defects. This software inspection methodology achieved the highest level to remove the defects efficiently and improves software quality.

There are some factors that generate the high quality software.

(i) The Phrases Design Inspection and Code Inspections – This factor refer to the formal inspections which follows the protocols such as training of participants, careful selection of participants, materials delivered for the inspection. Both moderator and recorder are present to analyze the defect statistics.

(ii) The Phrase Quality Assurance – This factor refers to an active software quality assurance group, which indulge in group of software developers to support them in developing the high quality software.

(iii) Formal Testing – It go threw the testing process under some conditions -

(a) For an application, a test plan was created.

(b) There are complete specification so that test cases can be created without notable gaps.

(c) Test library control tools are utilized.

(d) .Test coverage analysis tools are utilized.

Software Inspection Process – The inspection process was developed in mid 1970's, later it was extended and modified. The process should have entry criteria that determine if the inspection process is ready to begin. This prevents unfinished work products from entering the inspection process. The entry criteria may be a checklish including the items such as "The spell-checking of the document".

There are some of the stages in the software inspection process such as –

(i) Planning – The moderator plan the inspection.

(ii) Overview Meeting – The background of the work product is described by the author.

(iii) Preparation – The examination of the work product is done by inspector to identify the possible defects.

(iv) Inspection Meeting – The reader reads the work product part by part during this meeting and the inspectors point out the defects for every part.

(v) Rework – After the inspection meeting, the author makes changes to the work product according to the action plans.

(vi) Follow-up – The changes done by the author are checked to make sure that everything is correct.

Q.24. Explain the formal technical review (FTR). (R.G.P.V., June 2011)

Ans. A formal technical review is a software quality assurance activity performed by software engineers. The motives of FTR are,

(i) To uncover errors in function, logic, or implementation for any representation of the software.

(ii) To verify that the software under review meets its requirements.

(iii) To achieve software that is developed in a uniform manner.

(iv) To ensure that the software has been represented according to predefined standards; and

(v) To make projects more manageable.

Moreover, FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation. The FTR also serves to promote back up and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen.

Actually, the FTR is a class of reviews that includes walkthroughs, inspections, round-robin reviews and other small group technical assessments of software. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended. Here, some guidelines similar to those for a walkthrough are presented as a representative formal technical review.

The Review Meeting – Regardless of the FTR format that is chosen, each review meeting should abide by the following constraints –

(i) Between three and five people (typically) should be involved in the review.

(ii) Advance preparation should occur but should require no more than two hours of work for each person.

(iii) The duration of the review meeting should be less than two hours.

At the end of the review, all attendees of the FTR must decide whether to

(i) Accept the product without further modification.

(ii) Reject the product due to severe errors (once corrected, another review must be performed), or

(iii) Accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required).

The decision made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings.

Review Reporting and Record Keeping – During the FTR, a reviewer (the recorder) actively records all issues that have been raised.

These are summarized at the end of the review meeting and a review issues list is produced. Moreover, a formal technical review summary report is completed.

A *review summary report* answers three questions –

(i) What was reviewed?

(ii) Who reviewed it?

(iii) What were the findings and conclusions?

The *review issues list* serves two purposes.

(i) To identify problem areas within the product and

(ii) To serve as an action item checklist that guides the producer as corrections are made. An issue list is normally attached to the summary report.

Review Guidelines – Guidelines for the conduct of formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed. A review that is uncontrolled can often be worse that no review at all. The following represents a minimum set of guidelines for formal technical reviews –

(i) Review the product, not the producer.

(ii) Set an agenda and maintain it.

(iii) Limit debate and rebuttal.

(iv) Enunciate problem areas, but don't attempt to solve every problem noted.

(v) Take written notes.

(vi) Limit the number of participants and insist upon advance preparation.

(vii) Develop a checklist for each product that is likely to be reviewed.

(viii)Allocate resources and schedule time for FTRs.

(ix) Conduct meaningful training for all reviewers.

(x) Review your early reviews.

Because many variables (e.g., number of participants, type of work products, timing and length, specific review approach) have an impact on a successful review, a software organization should experiment to find what approach works best in a local context.

Q.25. Write short notes on –

(i) Black-box testing (ii) White-box testing. (R.G.P.V., June 2016)

Or

Explain black-box testing and white-box testing. (R.G.P.V., Dec. 2017)

Ans. (i) Black-box Testing – Functional or black-box testing is an approach to testing where the tests are derived on the basis of the requirements or specifications of the program or module and the internals of the module or the program are not considered.

Functional testing refers to testing which involves only observation of the output for some input values and there is no attempt to analyse the code, which produces the output. The internal structure of the program is ignored. Fig. 4.6 shows the model of a system which is assumed in black-box testing.

This approach can be applied to systems that are organized as functions or as objects. The tester gives inputs to the component or the system and checks the corresponding outputs. If the outputs are not satisfactory, then the test has successfully detected a problem with the software.

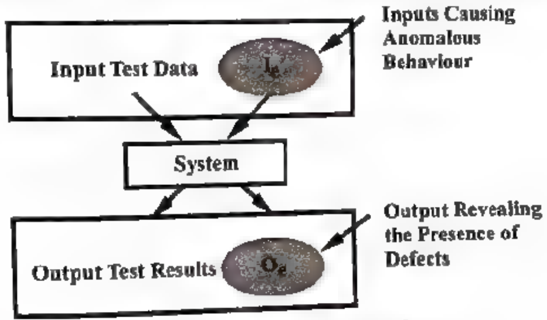

Input Test Data
I_e
Inputs Causing Anomalous Behaviour
System
Output Test Results
O_e
Output Revealing the Presence of Defects

Fig. 4.6 Black-box Testing

Advantages –

(a) Tester requires no knowledge of implementation and programming language used.

(b) Reveals any ambiguities and inconsistencies in the functional specifications.

(c) Efficient when used on larger systems.

(d) A non-technical person can also perform black box testing.

Disadvantages –

(a) Only a small number of possible inputs can be tested as testing every possible input consumes a lot of time.

(b) There can be unnecessary repetition of test inputs if the tester is not informed about the test cases that the software developer has already tried.

(c) Leaves many program paths untested.

(d) Cannot be directed towards specific segments of code, hence is more error prone.

(ii) White-box Testing – Structural testing is an approach to testing where the test cases are generated on the basis of the actual code of the program or module to be tested. This approach is also known as ***white-box*** testing, ***glass-box*** testing or ***clear-box*** testing.

Fig. 4.7 show the model of a system which is assumed in white-box testing.

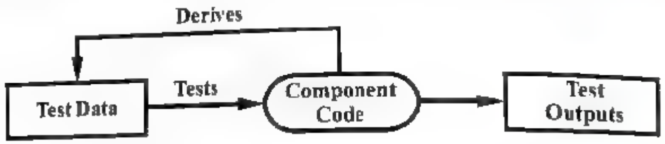


Fig. 4.7 Structural Testing

In structural testing, test group must have complete knowledge about the internal structure of the software. This approach is applicable to relatively small program units like subroutine, or the operations associated with an object. The tester can analyze the code and use knowledge about the structure of a component to generate test data. The code analysis can be used to find how many test cases are required to ensure that all of the statements in the program are run at least once during the testing process.

Advantages –

(a) Covers the larger part of the program code while testing.

(b) Uncovers typographical errors.

(c) Detects design errors that occur when incorrect assumptions are made about execution paths.

Disadvantages –

(a) Tests that cover most of the program code may not be good for assessing the functionality of surprise (unexpected) behaviours and other testing goals.

(b) Tests based on design may miss other system problems.

(c) Tests cases need to be changed if implementation changes.

Q.26. Explain the black-box testing. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.25 (i).

Q.27. Explain white-box testing. Discuss the advantages of mutation testing over control flow based and data flow based testing. ***(R.G.P.V., Dec. 2010)***

***Ans.* White-box Testing** – Refer to Q.25 (ii).

Advantage of Mutation Testing – An important advantage of mutation testing is that ıt can be automated to a great extent. By predefining a set of prımitive changes that can be applied to the program, the process of generating and killing of mutants can be automed. These primitive changes can be simple program alterations like changing the type of arithmetic operator, changing a logic operator, change the data type of a variable definition, changing the value of a constant, deleting a variable definition, deleting a statement, etc. Mutation testing performs quite well compared to other method of testing and increases the confidence in testing. It has been used to compare the test cases generated by different test case generation schemes – the mutation score is used as the metric for comparison, the higher the mutation score, the better the test case set or the test case generation strategy.

Q.28. Compare functional and structural testing approaches.
(R.G.P.V., Dec. 2016)

Or

Differentiate between black-box and white-box testings.
(R.G.P.V., Dec. 2003, June 2011, 2013)

Or

Write the difference between black-box testing and white-box testing.
(R.G.P.V., June 2014)

Or

Discuss the difference between black-box and white-box testing models.
(R.G.P.V., May 2019)

Ans. Following are the differences between structural and functional testing –

S.No.	*Black-box Testing*	*White-box Testing*
(i)	It is also known as ***functional testing*** or ***behavioural testing.***	It is also known as ***glass-box testing*** or ***structural testing.***
(ii)	It performs tests at software interface.	It performs close examination of procedural details.
(iii)	They test the operationability of software functions, acceptance of input and delivery of output and the maintainability of the external information integrity.	They test the software's logical path by having test cases exercising specific sets of conditions and loops.
(iv)	It examines some of the basic aspects of a system, having little regard for the internal logical software structure.	It examines the program status at various points to determine whether the expected status corresponds to the actual one.

Q.29. Explain equivalence partitioning technique of black-box testing. ***(R.G.P.V., June 2007)***

Or

What is equivalence partitioning ? ***(R.G.P.V., June 2016)***

Ans. This method tests the validity of outputs by partitioning the input domain into different classes of data, i.e. equivalence classes, using which test cases can be easily generated. Test cases are designed with the aim of covering each partition at least once. If a test case is able to detect all the errors in the specified partition, then the test case is called an ideal test case.

An equivalence class shows valid or invalid states for the input condition. An input condition can be either a specific numeric value, a range of values, a Boolean condition, or a set of values. The following are the general guidelines for generating the equivalence classes –

(i) If input consists of a specific numeric value, there will be three equivalence classes – one valid and two invalid classes.

(ii) If an input condition is Boolean, there will be two equivalence classes – one valid and one invalid class.

(iii) If input consists of a range, there will be three equivalence classes – one valid and two invalid classes.

(iv) If an input condition specifies a member of a set, there will be one valid and one invalid equivalence class.

Q.30. Explain the term boundary value analysis with suitable example. ***(R.G.P.V., June 2015)***

Or

Explain the boundary value analysis testing technique with the help of an example. ***(R.G.P.V., Dec. 2015)***

Or

What do you mean by boundary value analysis ? Give two examples of boundary value testing. ***(R.G.P.V., May 2019)***

Ans. Boundary value analysis (BVA) is a black box test design technique where test cases are designed on the basis of boundary values. Boundary value is defined as an input value or output value, which is at the edge of an equivalence partition or at the smallest incremental distance on either side of an edge. For example, the minimum or maximum value of a range.

BVA is used because it has been found that several errors occur at the boundary of the given input domain instead of the middle of the input domain.

For example, determine the boundary value test suite for a function that computes the square root of the integer values in the range of 0 and 5000, determine the boundary value test suite. Here, there are three equivalence classes – The set of negative integers, the set of integers in the range of 0 and 5000, and the set of integers larger than 5000. The boundary value based test suite is {0, -1, 5000, 5001}.

Following are the guidelines for BVA –

(i) If an input condition specifies a number of values, then test cases are generated to exercise the minimum and maximum numbers and values just above and below these limits.

(ii) If an input consists of certain data structures, then the test case should be able to execute all the values present at the boundaries of the data structures, like the maximum and minimum value of an array.

(iii) If an input consists of a range of certain values, then test cases should be able to exercise both the values at the boundaries of the range and the values that are just above and below boundary values.

(iv) If an input consists of a list of numbers, then the test case should be able to exercise the first and the last elements of the list.

Q.31. Explain orthogonal array testing method of black-box testing.

Ans. Orthogonal array testing is used in problems where the input domain is relatively limited i.e., the number of input parameters is small and their values are clearly bounded. This testing method is particularly useful in finding errors related with ***region faults***, which is an error category related with faulty logic within a software component.

The orthogonal array testing provides good test coverage with much fewer test cases compared to exhaustive approach. An L9 orthogonal array of the fax send function is shown in table 4.1.

Table 4.1 An L9 Orthogonal Array

Test Case	*Test Parameters*			
	P1	*P2*	*P3*	*P4*
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 4.1 shows the use of the L9 orthogonal array by a fax machine for the "send" function. The parameters, P1, P2, P3, and P4, are passed to the

send function, each of which takes three discrete values. For example, the values for P1 are as follows –

P1 = 1; send it now

P1 = 2; send it after an hour

P1 = 3; send it after midnight.

Similarly, P2, P3 and P4 may have values of 1, 2 and 3, signifying other send functions.

Q.32. What is the weakness found in equivalence partitioning and boundary value analysis methods ? How can cause-effect graphing overcome it ?

Ans. One weakness or limitation of equivalence partitioning and boundary value analysis methods is that they both consider each input separately i.e., both focus on the conditions and classes of one input. They do not take into account the combinations of input circumstances that may form interesting situations to be tested. One simple way to exercise combinations of several different input conditions is to consider all valid equivalence classes combinations of input conditions. This approach will lead to an unusually large number of test cases, several of them will not be useful for revealing any new errors. For example, if we have n different input conditions, such that any of their combinations is valid, we will have 2^n test cases.

Cause-effect graphing is a technique that aids in choosing combinations of input conditions in a systematic way, so that the number of test cases does not become usually large. The first step in this technique is to identify causes and effects of the system under testing. (A cause is a distinct input condition, and an effect is distinct output condition). Each of the two conditions forms a node in the cause effect graph. The conditions should be stated in such a way that they can be set to either true or false. For example, an input condition can be "file is full", which can be set to true by having a completely filled input file, and false by an empty or partially filled file. After having the causes and effects identified, we determine, for each effect, the causes that can produce that effect and how the conditions should be combined to make the effect true. The conditions are combined by having the Boolean operators "AND", "OR", and "NOT". They are represented in the graph by "&", "|", and "~", respectively. Now for each effect, all the causes combinations, that will make the depending effect true, are generated. Thus, we can then identify the combinations of conditions making different effects true. Finally, a test case is generated for each combination of conditions, which make some effect true.

Cause-effect graphing, in addition with generating high-yield test cases, aids the understanding of the system functionality, since the tester must identify the distinct causes and effects.

Q.33. Define functional testing. Also, explain the various approaches used in functional testing.
(R.G.P.V., June 2015)

Ans. **Functional Testing** – Refer Q.25 (i).

Various Approaches – Refer Q.29, Q.30, Q.31 and Q.32.

Q.34. Discuss about path testing.

Ans. Path testing is a structural testing technique. It's purpose is to exercise every independent execution path through a component or program. If each independent path is run, then all the component statements must have been run at least once. Moreover, all the conditional statements are tested for both true and false cases.

The number of paths through a program is proportional to its size. With the increase of modules integrated into systems, it becomes unmanageable to use structural testing techniques. Therefore, path testing techniques are often used at the unit testing and module testing stages of the testing process.

All possible combinations of all paths through the program are not tested by path testing. This is an impossible objective for any components apart from very trivial one without loops. An infinite number of possible path combinations exist in programs with loops. Defects showing themselves when specific path combinations arise may still exist, even though all program statements have been run (at least once).

Path testing starts with a program flow graph, which is a skeleton of a program through which all paths flow.

A flow graph contains nodes and edges representing decisions and control flow, respectively. The flow graph can be built by replacing program control statements by equivalent diagrams. A program having no go to statements can be simply converted to its flow graph. Sequential statements, such as assignments, procedure calls, and I/O statements, can be ignored in building a flow graph. Each branch in a conditional statement is represented as a separate path, and loops are shown by an arrow, looping back to the loop condition node.

The aim of structural testing is to ensure that each independent program path is run at least once. An independent program path traverses at least one new edge in the flow graph. In terms of program, it means exercising one or more new conditions. All conditions must be executed for Both the true and false branches.

Q.35. Write a detailed note on control structure testing.
(R.G.P.V., Dec. 2005, June 2006)

Ans. Control structure testing is used to increase the coverage area by testing various control structures present in the program. The different types of testing performed under control structure testing are as follows –

(i) Condition Testing – Condition testing is a test cased design method, which ensures that the logical conditions and decision statements are free from errors. The errors present in logical conditions can be incorrect Boolean operators, missing parenthesis in a Booleans expression, error in relational operators, arithmetic expressions, and so on.

The common types of logical conditions that are tested using condition testing are –

(a) A relation expression, like E1 op E2 where 'E1' and 'E2' are arithmetic expressions and 'op' is an operator.

(b) A simple condition like any relational expression preceded by a NOT (~) operator. For example, (~ E1) where 'E1' is an arithmetic expression and 'a' denotes NOT operator.

(c) A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis. For example, (E1 & E2) | (E2 & E3) where E1, E2, E3 denote arithmetic expression and '&' and '|' denote AND or OR operators.

(d) A Boolean expression consists of operands and a Boolean operator like 'AND', OR, NOT. For example, 'A|B' is a Boolean expression, where 'A' and 'B' denote operands and | denotes OR operator.

(ii) Data Flow Testing – The data flow testing method chooses test paths of a program on the basis of the locations of definitions and uses of variables in the program.

The data flow testing approach is illustrated as follows – suppose that each statement in a program is given a unique statement number and that each function can not modify its parameters or global variables. For a statement with S as its statement number.

DEF (S) = {X | statement S has a definition of X}

USE (S) = {X | statement S has a use of X}

If statement S is an if or loop statement, its DEF set is empty and its USE set depends on the condition of statement S. The definition of variable X at statement S is called to be *line* at statement S' if there is a path from statement S to statement S' that has no other definition of X.

A definition-use (DU) chain of variable X has the form [X, S, S'], where S and S' denote statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is line at statement S'.

A simple data flow testing approach is to require that each DU chain be covered at least once. This approach is referred to as the DU testing approach. The DU testing does not ensure the coverage of all branches of a program. However, a branch is not guaranteed to be covered by DU testing only in rare cases like if-then-else constructs in which the then part contains no definition of any variable and the else part does not exist.

Data flow testing strategies are appropriate for choosing test paths of a program containing nested if and loop statements.

(iii) Loop Testing – Loop testing is actually a white-box testing technique. It focuses exclusively on the validity of loop constructs. Following are the types of loops.

(a) Simple Loops – The following set of tests can be applied for simple loops, where the maximum number of allowable passes through the loop is n.

(1) Skip the entire loop.

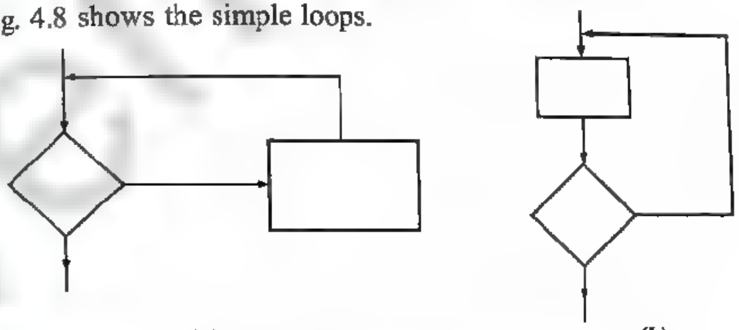
(2) Traverse the loop only once.

(3) Traverse the loop two times.

(4) Make p passes through the loop where p < n.

(5) Traverse the loop n – 1, n, n + 1 times.

Fig. 4.8 shows the simple loops.



(a) *(b)*

Fig. 4.8 Simple Loops

(b) Concatenated Loops – If loops do not depend on each other, then concatenated loops can be tested using the approach used in simple loops. If the loops are dependent on each other, then steps in nested loops are followed (see fig. 4.9).

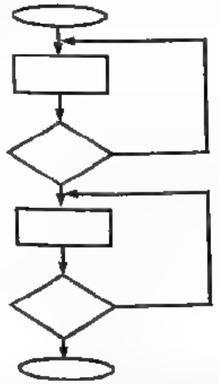


Fig. 4.9 Concatenated Loops

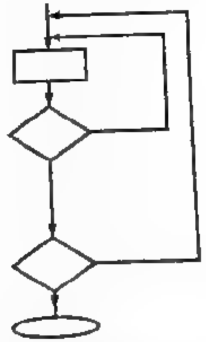


Fig. 4.10 Nested Loops

(c) Nested Loops – Loops within loops are called as nested loops. While testing nested loops, the number of tests increases as the level of nesting increases. The steps followed for testing nested loops are as follows –

(1) Start with the inner loop and set values of all the outer loops to minimum.

(2) Test the inner loop using the steps followed for testing simple loops while holding the outer loops at their minimum parameter values. Add other tests for values that are either out-of-range or are eliminated.

(3) Move outwards, conducting tests for the next loop. However, keep the nested loops to typical values and outer loops at their minimum values.

(4) Continue testing until all loops are tested.

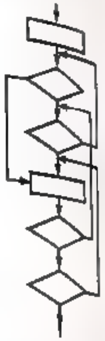


Fig. 4.11 Unstructured Loops

(d) Unstructured Loops – This type of loops should be redesigned, whenever possible, to reflect the use of the structured programming constructs (see fig. 4.11).

Q.36. Explain various approaches of testing and also explain their testing. ***(R.G.P.V., Dec. 2006, June 2012)***

Ans. There are two approaches of testing -

(i) Functional testing (ii) Structural testing.

(i) Functional Testing – Refer to Q.25 (i).

Functional testing covers the following types of testing methods –

(a) Equivalence Partitioning – Refer to Q.29.

(b) Boundary Value Analysis – Refer to Q.30.

(c) Orthogonal Array Testing – Refer to Q.31.

(d) Cause-effect Graphing – Refer to Q.32.

(ii) Structural Testing – Refer to Q.25 (ii).

Structural testing covers the following types of testing methods –

(a) Basic Path Testing – Refer to Q.34.

(b) Condition Testing – Refer to Q.35 (i).

(c) Data Flow Testing – Refer to Q.35 (ii).

(d) Loop Testing – Refer to Q.35 (iii).

Q.37. Do you agree with the following statement. "System testing can be considered a pure black-box test" ? Justify your answer. ***(R.G.P.V., June 2014)***

Ans. Performance testing is an important type of system testing. There are several types of performance testing corresponding to various types of nonfunctional requirements. For a particular system, the types of performance testing to be carried out on a system depends on the different non-functional requirements of the system documented in its SRS document. All performance tests can be considered as black-box tests.

Q.38. Write short note on unit testing. ***(R.G.P.V., June 2016)***

Ans. Unit testing concentrates the verification effort on the smallest unit of software design i.e., the software component or module. It uses the component-level design description as a guide to test major control paths to uncover errors within the module boundary. It is white-box oriented and the step can be performed in parallel for multiple components.

Unit testing is a dynamic method for verification, where the program is actually compiled and executed. It is one of the most widely used methods, and the coding phase is sometimes called "coding and unit testing phase". Similar to the other testing methods, unit testing involves executing the code with some test-cases and then evaluating the results.

The goal of unit testing is to test modules or "units", not the whole software system. Unit testing is most often done by the programmer himself. The programmer, after having done the coding of a module, tests it with some test data. This tested module is then delivered for system integration and further testing.

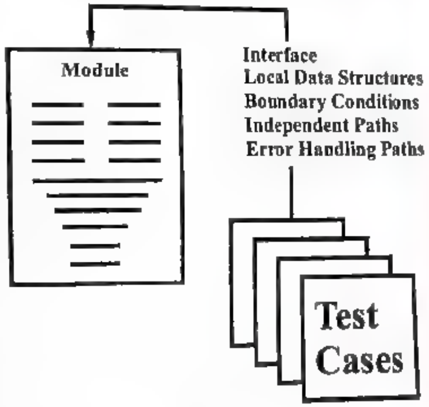


Fig. 4.12 Unit Test

The tests that occur during unit tests are shown in fig. 4.12.

The "module interface" is tested to make sure information flow in and out of the program unit under consideration. Then the "local data structures" are tested to make sure that data stored temporarily maintain their integrity during all steps in an algorithm's execution. The "boundary conditions" are tested to make sure that the module operates properly at boundaries. All "independent paths" through the control structure are checked to make sure that all statements in a module have been executed. Finally, all "errors handling paths" are tested.

Data flow tests across a module interface are needed before initiating any other test. Also local data structures should be checked and the local impact on global data should be confirmed, while unit testing.

Q.39. What are the types of unit testing ?

Ans. The following types of unit testing shown in fig. 4.13.

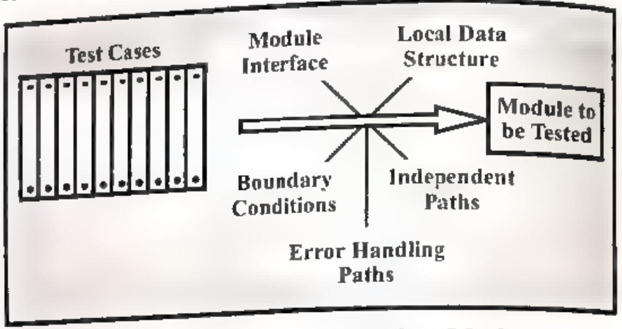


Fig. 4.13 Various Unit Testing Methods

(i) Module Interface – These are tested to ensure that information flows in a proper manner into and out of the 'unit' under test. The test of data flow (across a module interface) is required before any other test is initiated.

(ii) Local Data Structure – These are tested to ensure that the temporarily stored data maintains its integrity while an algorithm is being executed.

(iii) Boundary Conditions – These are tested to ensure that the module operates as desired within the specified boundaries.

(iv) All Independent Paths – These are tested to ensure that all statements in a module have been executed at least once. In this testing, the entire control structure should be exercised.

(v) Error-handling Paths – After successful completion of the various tests, error-handling paths are tested.

Q.40. Discuss unit test environment.

Or

Explain the driver and stub technique of unit testing taking a suitable example. (R.G.P.V., June 2006)

Ans. **Procedure** – Unit tests can be designed before coding begins or after the code is developed. Review of this design information guides the creation of test cases, which are used to detect errors in various units. Since a component is not an independent program, two modules, drivers, and stubs are used to test the units independently. Driver is a module that passes input to the unit to be tested. It accepts test case data and then passes the data to the unit being tested. Next, the driver prints the output produced. Stub is a module that works as a unit referenced by the unit being tested. It uses the interface of the subordinate unit, does minimum data manipulation and returns control back to the unit being tested shown in fig. 4.14.

Drivers and stubs imply overhead. It means that both are software that must be written but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. However, many components cannot be adequately unit tested with simple overhead software. In such situations, complete testing can be postponed until the integration test step.

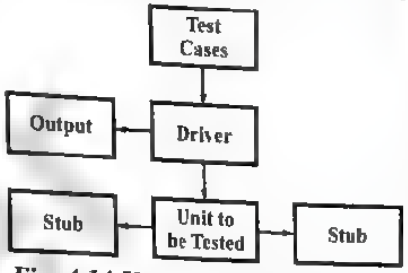


Fig. 4.14 Unit Testing Environment

Q.41. Explain the integration testing.(R.G.P.V., Dec. 2005, May 2018)

Or

What is integration testing technique ? How many types of it ? Explain. (R.G.P.V., Dec. 2015)

Or

Describe integration testing and what are the approaches for integration testing. (R.G.P.V., June 2016)

Or

What is integration testing ? Discuss about the various approaches of integration testing. (R.G.P.V., Dec. 2017)

Ans. Integration testing is a systematic technique to build the program structure and at the same time tests are performed to uncover errors related with interfacing. The objective of integration testing is to take unit tested components and construct a program structure that has been specified by design. The various integration testing strategies used are –

(i) Top-down integration testing (ii) Bottom-up integration testing
(iii) Regression testing (iv) Smoke testing.

(i) Top-down Integration Testing – It is an incremental approach for building program structure. As we move downward through the control hierarchy, modules are integrated (beginning with the main program at the top). Modules subordinate to the main control module are included into structure (by either depth-first or breadth-first manner).

Fig. 4.15 illustrates top-down integration in depth-first manner. Depth-first integration would integrate all components on a major control path of the structure. This major path is selected arbitrarily and depends on application-specific characteristics. For example, in fig. 4.15, A1, A2, A5 would be integrated first. Then, A8 or A6 would be integrated. After that, the central and right-hand control paths are constructed.

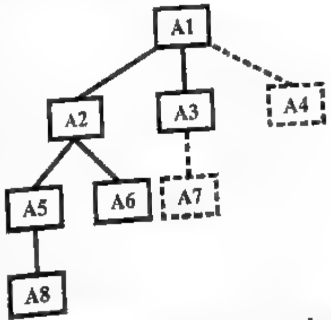


Fig. 4.15 Top-down Integration

On the other hand, breadth-first integration includes all components directly subordinate at each level, moving horizontally across the structure. Referring to the fig. 4.15, components A2, A3, and A4 would be integrated first. Next, A5, A6, and so on.

The necessary steps performing the integration process are as follows –

(a) The main control module or main program is used as a test driver and stubs are replaced for all components directly subordinate to the main program.

(b) Using either depth-first or breadth-first approach, subordinate stubs are substituted one at a time with actual components.

(c) Every time a row module is integrated, tests are conducted.

(d) After completing tests, another stub is substituted with the real component.

(e) Regression testing may also be conducted for avoiding new errors.

This process continues from step (b) until the entire program structure is constructed.

(ii) Bottom-up Integration Testing – Bottom-up integration testing starts building and testing with atomic modules i.e., components at the lowest levels in the program structure. Since in this approach, components are integrated from bottom towards up, processing needed for components subordinate to a given level is always available and the requirement for stubs is removed.

Following are the necessary steps for implementing bottom-up integration –

(a) The combination of low-level components results in clusters that perform a specific software subfunction.

(b) A driver i.e., a control program for testing, is written to help test case input and output.

(c) The cluster is tested.

(d) Drivers are then eliminated and clusters are merged moving up in the program structure.

Fig. 4.16 illustrates the bottom-up integration pattern. Observe that, components are merged to form clusters 1, 2, and 3. Each cluster is tested by a driver (shown by dashed box). Components in clusters 1 and 2 are subordinate to P_a. Drivers C1 and C2 are eliminated and the clusters are interfaced directly to P_a. In the same way, driver C3 for cluster 3 is eliminated before integrating with module P_b. Then, both P_a and P_b will finally be integrated with component P_c, and so on.

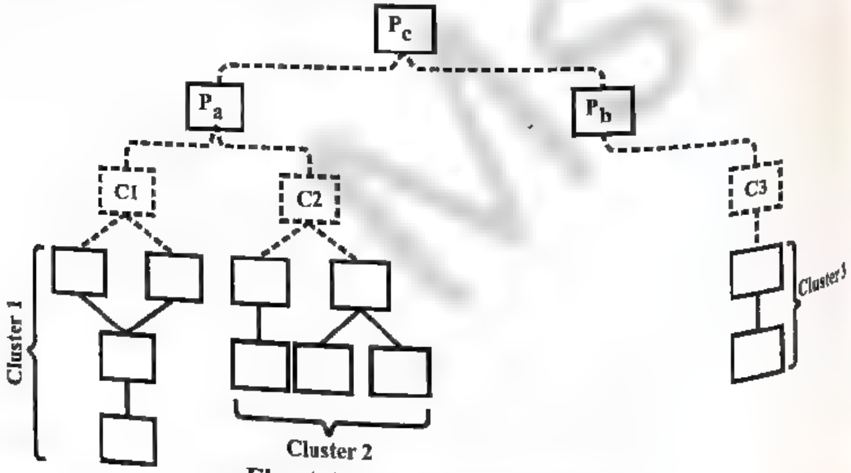


Fig. 4.16 Bottom-up Integration

Moving upwards, reduces the need for separate test drivers. In fact, the top-down integration of the top two levels of program structure can substantially reduces the number of drivers and simplifies clusters integration.

(iii) Regression Testing – Regression testing reexecutes some subset of tests that have already been performed to ensure that changes have not introduced unintended side effects. Precisely, "regression testing is an important strategy for reducing side effects".

Regression testing may be performed manually, by re-executing a subset of all test cases or using automated capture/playback tools which enable the tester to capture test cases and results for subsequent playback and comparison.

Following are the three different classes of test cases contained by the regression test suite i.e., the subset of tests to be executed

(a) Tests that are used to execute software functions.

(b) Additional tests concentrating on software functions which are likely to be affected by the change.

(c) Tests emphasizing on the changed software components.

With the integration testing, the regression tests can build quite large. Thus, the regression test suite should be designed such that to include only those tests that indicate one or more classes of errors in each major program function. It is quite impossible and practically inefficient to re-execute every test for every program function after each change.

(iv) Smoke Testing – Smoke testing is an integration testing strategy. It is used when "shrink-wrapped" software products are being developed. It is mainly used for time critical software and permits the development team to assess the software frequently. The activities involved in the smoke testing approach are as follows –

(a) The translated software components into code are integrated into a "build". A build contains all data files, libraries, reusable modules, and engineered components that are needed for the implementation of one or more product functions.

(b) A sequence of tests is designed to discover errors that will keep the build from performing its function properly. This intention is to uncover "show stopper" error that have the highest chances of throwing the software project behind schedule.

(c) The build is integrated with several other builds and the entire product is smoke tested every day. This may be done by using either top-down or bottom-up approach.

Following benefits are provided by smoke testing when applied on complex, time-critical software engineering projects –

(a) Minimizes the Integration Risk – As the smoke tests are performed daily, incompatibilities and other show-stopper errors are uncovered early, and hence decreasing the chances of serious schedule impact when errors are exposed.

(b) Improves Quality of End-product – As this is an constructive approach, smoke testing is likely to expose both functional errors and architectural and procedural design defects. Early correction of these errors improves the product quality.

(c) Simplifies Error Diagnosis and Correction – Similar to all integration testing strategies, errors uncovered during smoke testing may be associated with "new software increments" i.e., the software just added is probable cause of newly exposed error.

(d) Easier Assess of Progress – Everyday, more of the software has been integrated and demonstrated to work. This enhances the morale of the developer team and provides team managers a good indication that progress is being made.

Q.42. Difference between the top-down and bottom-up integration testing. ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.41 (i) and (ii).

Q.43. Explain integration testing. What are the steps for top-down integration ? What problems may be encountered when top-down integration is chosen ? ***(R.G.P.V., June 2008)***

Or

Describe integration testing. Explain the steps for top-down integration. Also list out shortcomings of top-down integration. (R.G.P.V., June 2015)

Ans. **Integration Testing and Steps for Top-down Integration** – Refer to Q.41.

Problems Encountered in Top-down Integration – Top-down integration strategy seems relatively uncomplicated, but practically, there may arise logistical problems. The major problem occurs when processing at low levels in the hierarchy is required to test the upper levels adequately. Stubs replace low-level modules at the beginning of top-down testing. Therefore, no data can flow upward in the program structure. The tester has three choices, as follows–

(i) Delaying many tests until replacing stubs with actual modules.

(ii) Developing stubs performing limited functions that simulate the actual module.

(iii) Integrating the software from the bottom to up in the hierarchy.

The first of the above noted approaches causes us to loose some control over correspondence between specific tests and inclusion of specific modules. This can cause problem in determining the cause of errors and tends to violate the highly constrained nature of the top-down approach. While, the second approach is workable but it can lead to significant overhead, as stubs become more and more complex. The third approach is known as bottom-up testing.

Q.44. What is system testing ? What series of tests are performed during system testing ? ***(R.G.P.V., Dec. 2008, June 2011, Dec. 2014)***

Or

What do you mean by system testing ? Explain in detail. ***(R.G.P.V., May 2019)***

Ans. ***System testing*** is actually a sequence of several different tests conducted to fully exercise the computer-based system. Although there is different purpose for each test, each test works to verify that system elements have been properly integrated and perform predetermined functions. The following are the types of system tests that are worthwhile for software-based systems –

(i) Recovery Testing – It is a system test forcing the software to fail in a number of ways, and verifies that recovery is well performed. If recovery is performed by the system itself, then reinitialization, check-pointing methods, data recovery, and restart are evaluated for correctness. And if recovery needs human intervention, the mean-time-to-repair i.e, MTTR is evaluated to know whether it is within acceptable limits.

(ii) Security Testing – It tries to verify that the protection mechanisms, provided in a system, will actually protect it from improper or illegal penetration. Penetration spans a broad range of activities as follows –

(a) Hackers trying to penetrate systems for sport.

(b) Disgruntled employees trying to penetrate for revenge.

(c) Dishonest individuals trying to penetrate for illicit personal gain.

The system's security must also be tested for invulnerability from frontal attack.

During security testing, the tester behaves as if he desires to penetrate the system. He may attempt to acquire passwords through external sources, or he may attack the system with custom software designed to destroy any defences, or he may overwhelm the system, thus denying service to others. He may also browse through insecure data, assuming to find the key to system entry, or he may intentionally cause system errors, hoping to penetrate during recovery.

Finally, a good security testing will lead to penetrate a system. But, the system designer will have to make penetration cost more than the value of information that will be obtained.

(iii) Stress Testing – It executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.

For example –

(a) Special tests may be developed that provide ten interrupts per second, when one or two is the average rate.

(b) Input data rates may be enhanced by an order of magnitude determining the input function response.

(c) Test cases likely to cause thrashing in a virtual operating system are designed.

(d) Test cases requiring maximum memory or other resources are executed.

(e) Test cases likely to cause excessive hunting for disk-resident data are created.

A little variation of stress testing is known as ***sensitivity testing*** technique, which attempts to uncover data combinations within valid input classes, causing instability or improper processing.

(iv) Performance Testing – It is designed to test the run-time performance of software within the context of an integrated system. Performance testing takes place throughout all processing steps of testing. Even at the low or basic level, an individual module's performance may be ascertained as white-box tests are performed. However, it is not until all elements of system are fully integrated for the assurance of the true performance of a system.

Mostly, performance tests are coupled with stress testing. They need both hardware and software instrumentation. It means that it is often essential to know resource utilization in an exacting fashion.

Q.45. What do you understand by system testing ? What are the different kinds of system testing that are usually performed on large software products?

(R.G.P.V., June 2012, 2014)

Ans. **System Testing** – Refer to Q.44.

Alpha and Beta Testing – System testing is performed when software integration is over and successfully tested, and the software product is now ready for inspection from the customer's point of view.

Two tests are performed - alpha and beta. The alpha test is performed at the developer's site by the customer under the project team's guidance. Here, users test the software on the development platform and find out errors for correction. Since alpha test is conducted by a few users, it has limited ability to find out errors and correcting them. After alpha testing, the software product is ready for transition to the customer site for implementation and deployment.

If the system is complicated, the software is not taken for implementation directly. The software is installed and all users use the software under testing mode. This is known as the beta test. In other words, beta tests are conducted at customer site in an environment where the software is exposed to several users. When the software is in use, the developer may or may not be present. Here, users record errors and report them periodically. The developers correct all errors. So beta test is a real life software experience without actual implementation.

Q.46. Supposed a developed software has successfully passed all the three level of testing i.e., unit testing, integration testing and system testing can we claim that the software is defect free ? Justify your answer ?

(R.G.P.V., June 2014)

Ans. A software product is tested in three levels or stages – unit testing, integration testing and system testing. During unit testing, the individual components (or units) of a program are tested.

After testing all the units individually, the units are slowly integrated and tested after each step of integration (integration testing). Finally, the fully integrated system is tested (system testing). Integration and system testing are called as testing in the large.

Often beginners ask a question that why test each module (unit) in isolation first, then integrate these modules and test, and again test the integrated set of modules – why not just test the integrated set of modules once thoroughly ? The answer to this question is there are two main reasons to it. First, while testing a module, other modules with which this module needs to interface may not be ready. Moreover, it is always a good idea to first test the module in isolation before integration because it makes debugging easier. If a failure is detected when an integrated set of modules is being tested, it would be hard to ascertain which module exactly has the error.

Q.47. Explain validation testing. ***(R.G.P.V., Dec. 2005)***

Ans. At the end of integration testing, software is assembled as a package having all the errors uncovered and corrected. At this time, a final series of software tests may start. It is called ***validation testing***. Validation can be defined simply as validation succeeds when software functions in a reasonably expected manner by the customer.

Software validation is obtained through a sequence of black-box tests demonstrating conformity with requirements. A test plan outlines the classes of tests to be performed and a test procedure specifies test cases that will be used for demonstrating confirmity with requirements. Both the plan and procedure are designed for the surity of functional, behavioural, and performance requirements. Also for ensuring that documentation is correct and human-engineered and other requirements are satisfied i.e., transportability, compatibility, error recovery, and maintainability.

After performing each validation test case, one of the following two conditions exist –

(i) The functional or performance characteristics conform to specification and are accepted.

(ii) A deviation from specification is exposed and a deficiency list is made.

A major element of the validation process is a configuration review, which is conducted to ensure that all software configuration elements have been well-developed, well-cataloged, and have the essential detail to bolster the support phase of the software life cycle.

Q.48. Explain acceptance testing.

Ans. Acceptance testing is the last step in the testing process, before providing the system to the customer for its operational use. It is performed with data given by system customer instead of simulated test data. Acceptance testing may uncover errors and omissions in the system requirements definition because the actual data exercise the system in another ways from the test data. It also uncovers requirements problems where the system's facilities do not satisfy the customer's need or the performance of the system is unacceptable.

Acceptance testing is also sometimes known as alpha testing. Bespoke systems are developed for a single customer. The alpha testing continues until the system developer and the customer agree that the provided system is an acceptable implementation of the system requirements.

On the other hand, when a system is to be marketed as a software product, another process known as beta testing is often performed. During beta testing, a system is delivered among several users who want to use it. The customers then convey problems to the developers. This provides the product for real use and detects errors which may not have been expected by the system developers. Consequently software engineer make modifications and then prepare for release of the software product to the whole customer base.

Q.49. What is cyclomatic complexity ? How is it useful in program testing ? ***(R.G.P.V., Dec. 2010)***

Or

How can you compute the cyclomatic complexity of a program ? How is it useful in program testing ? ***(R.G.P.V., June 2013)***

Ans. Cyclomatic complexity is defined as software metric that provides a quantitative measure of the logical complexity of a program. The number of independent paths present in the program is calculated using cyclomatic complexity. This software metric also provides information about the number of tests needed to ensure that all statements in the program are executed at least once.

A path through the program, which specifies a new condition or a minimum of one new set of processing statements, is known as an independent path. In other words, an independent path must move along at least one edge that has not been traversed before the path is defined. The flow graph for the flowchart given in fig. 4.17 is shown in fig. 4.18. The independent paths for the flow graph shown in fig. 4.18 are

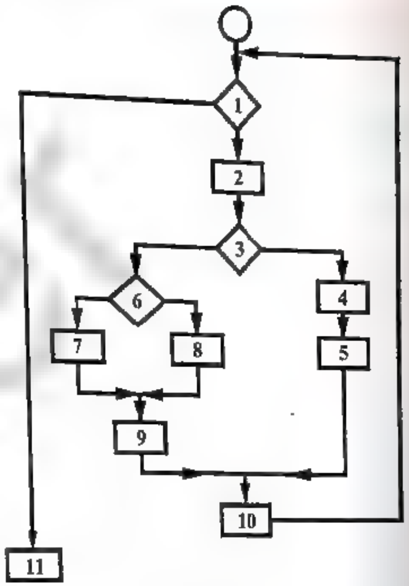


Fig. 4.17

Path 1 : 1 – 11

Path 2 : 1 – 2 – 3 – 4 – 5 – 10 – 1 – 11

Path 3 : 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11

Path 4 : 1 – 2 – 3 – 6 – 7 – 9 – 10 – 1 – 11

Here notice that, each new path introduces a new edge. The path

1 – 2 – 3 – 4 – 5 – 10 – 1 – 2 – 3 6 – 8 – 9 – 10 – 1 – 11

is not an independent path as it is just a combination of already specified paths and does not traverse any new edges.

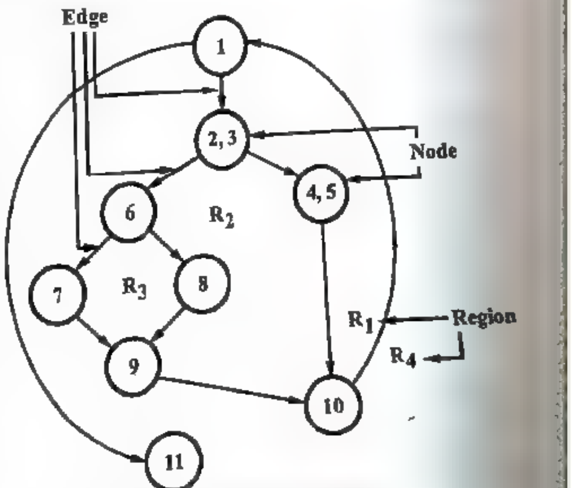


Fig. 4.18 Flow Graph

Paths 1, 2, 3 and 4 make a ***basis set*** for the flow graph in fig. 4.18. It means that if tests are designed to force execution of these paths (a basis set), each statement in the program will have been guaranteed to be run at least one time and every condition will have been run on its true and false sides. The basis set is not unique, but several basis sets can be derived for a given procedural design.

Cyclomatic complexity can be calculated by using any of the three methods given below –

(i) The total number of regions present in the flow graph of a program represents the cyclomatic complexity of the program.

(ii) Cyclomatic complexity, V(G), for a flow graph, G, can be calculated as

V(G) = E – N + 2

where E denotes the number of flow graph edges, N denotes the number of flow graph nodes.

(iii) Cyclomatic complexity, V(G), for a flow graph, G, can be calculated as

V(G) = P + 1

where, P denotes the number of predicate nodes contained in the flow graph G.

The cyclomatic complexity calculated for the flow chart shown in fig. 4.17, is shown below –

(i) The flow graph contains four regions.

(ii) V(G) = 11 edges – 9 nodes + 2 = 4

(iii) V(G) = 3 predicate nodes + 1 = 4

Hence, the cyclomatic complexity for the above flowgraph is 4.

Q.50. Draw a flow graph for binary search, find all independent paths.

(R.G.P.V., June 2005)

Ans. A flow graph for a binary search routine is shown in fig. 4.19.

By tracing the flow, we find that the independent paths through this binary search flow graph are as follows –

a, b, c, h, i

a, b, c, d, f, g, b

a, b, c, d, e, g, b

a, b, c, d, f, g, b, h, i.

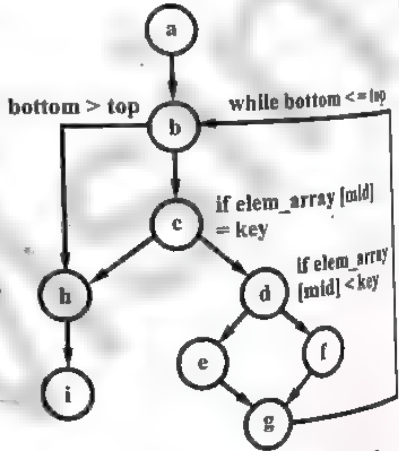


Fig. 4.19 Flow Graph for a Binary Search Routine

Q.51. What is test plan ? Write down the components of test plan and their purpose.

(R.G.P.V., June 2012)

Ans. A test plan describes how testing would be accomplished. A test plan is defined as a document that describes the objectives, scope, method and purpose of software testing. This plan identifies test items, features to be tested, testing tasks, and the persons involved in performing these tasks. It also identifies the test environment and the test design and measurement techniques that are to be used.

A complete test plan helps people outside the test group to understand the "why" and 'how' of product validation. An incomplete test plan can result in a failure to check how the software works on different hardware and operating systems or when the software is used with other software. To avoid this problem, IEEE states some components that should be covered in a test plan. The test plan components are shown in table 4.2.

Table 4.2 Test Plan Components

S.No.	*Component*	*Purpose*
(i)	Responsibilities	Assigns responsibilities and keeps people on track and focused.
(ii)	Assumptions	Avoids misunderstandings about schedules.
(iii)	Test	Outlines the entire process and maps specific tests. The testing scope, schedule, and duration are also outlined.
(iv)	Communication	Communication plan (who, what, when, how about the people) is developed.
(v)	Risk Analysis	Identifies areas that are critical for success.
(vi)	Defect Reporting	Specifies how to document a defect so that it can be reproduced, fixed, and retested.
(vii)	Environment	Specifies the technical environment, data, work area, and interfaces used in testing. This reduces or eliminates misunderstandings and sources of potential delay.

Q.52. Write the steps to development of a test plan.

Ans. To develop a test plan, the following steps are followed as shown in fig. 4.20.

(i) Set Objectives of Test Plan – Before developing a test plan, it is necessary to understand its purpose. The objectives of a test plan depend on the objectives of a software. For example, if the objective of a software is to accomplish all user requirements, then a test plan is generated to meet this objective. Thus, it is necessary to determine the objectives of a software before identifying the objective of the test plan.

(ii) Develop a Test Matrix – A test matrix indicates the components of a software that are to be tested. It also specifies the test required to check these components. Test matrix is also used as a test proof to show that a test exists for all components of a software that require testing. In addition, test matrix is used to indicate the testing method which is used to test the entire software.

(iii) Develop Test Administrative Component – It is necessary to prepare a test plan within a fixed time so that software testing can begin as soon as possible. It specifies the time schedule and resources (administrative people involved while developing the test plan) required to execute the test plan. However, if the implementation plan of software changes, the test plan also changes. In this case, the schedule to execute the test plan also gets affected.

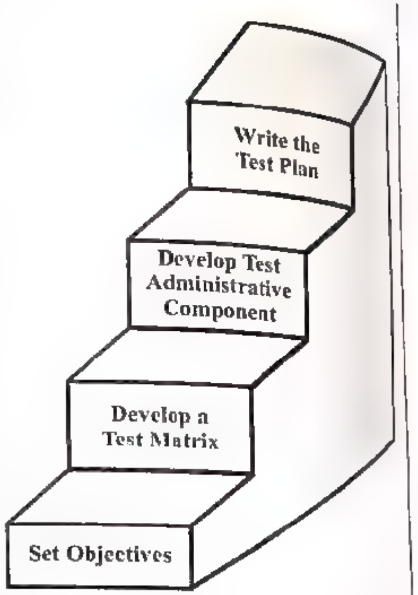


Fig. 4.20 Steps in Test Plan

(iv) Write the Test Plan – The components of a test plan, such as its objectives, test matrix and administrative component are documented. All these documents are then collected together to form a complete test plan. These documents are organized either in an informal or formal manner. In the informal manner, all the documents are collected and kept together. The testers read all the documents to extract information required for testing the software. On the other hand, in a formal manner, the important points are extracted from the documents and kept together. This makes it easy for testers to extract important information, which they require during software testing.

Q.53. Describe the test metrics. ***(R.G.P.V., Dec. 2016)***

Or

Explain the test metrics. ***(R.G.P.V., May 2018)***

Ans. Metrics are simply measurements. Test metrics are those measurements from your test process that will help you determine where the application stands and when it will be ready for release. In an ideal world, you would measure your tests at every phase of the development cycle, thus gaining an objective and accurate view of how through your tests are and how closely the application compiles with its requirements.

Software testing metrics provide quantitative approach to measure the quality and effective of the software development and testing process. It helps the team to keep a track on the software quality at every stage in the software development cycle and also provides information to control and reduce the number of errors. It allows the stakeholders to measure the efficiency of the team and accelerates application delivery.

Q.54. Discuss the types of metrics for testing.

Ans. Testing metrics fall into two broad categories – (i) metrics that attempt to predict the likely number of tests required at various testing levels and (ii) metrics that focus on test coverage for a given component.

Function-based metrics are used as a predictor for overall testing effort. A number of project-level characteristics like testing effort and time, errors uncovered, number of test cases produced, for past projects can be gathered and correlated with the number of FP produced by a project team. The team can then project expected values of these characteristics for the current project.

The bang metric can give an indication of the number of test cases needed by looking at the primitive measures. The number of functional primitives (FuP), data elements (DE), objects (OB), relationships (RE), states (ST), and transitions (TR) are used to project the number and types of black-box and white-box tests for the software. For instance, the number of tests related with the human/computer interface can be estimated by (i) looking at the number of transitions (TR) contained in the state transition representation of the HCI and evaluating the tests needed to exercise each transition, (ii) looking at the number of data objects (OB) that move across the interface, and (iii) the number of data elements that are input or output.

The information on the ease or difficulty related with integration testing and the need for specialized testing software (e.g., stubs and drivers) is provided by the architectural design metrics. Cyclomatic complexity lies at the core of basis path testing. Furthermore, cyclomatic complexity are used to target modules as candidates for extensive unit testing. High cyclomatic complexity modules are more likely to be error prone than low cyclomatic complexity modules. That is why, the tester should expend above average effort to expose errors in such modules before integrating them in a system. The estimation of testing effort can also be done using metrics derived from Halstead measures. Using the definitions for program volume V, and program level PL, software science effort e, can be computed as

$$PL = 1/[(n_1/2) \times (N_2/2)] \quad ...(i)$$

$$e = V/PL \quad ...(ii)$$

The estimation of the percentage of overall testing effort to be allocated to a module k can be found using the following relationship –

$$\text{Percentage of testing effort } (k) = e(k)/\Sigma e(i) \quad ...(iii)$$

where e(k) is computed for module k using equations (i) and (ii) and the summation in the denominator of equation (iii) is the sum of software science effort across all modules of the system.

As tests are performed, three different measures give an indication of testing completeness. A measure of the breath of testing gives an indication of how many requirements have been tested. This gives an indication of the completeness of the test plan. A measure of the percentage of independent basis paths covered by testing versus the total number of basis paths in the program is depth of testing. A reasonably accurate estimate of the number of basis paths can be calculated by adding the cyclomatic complexity of all program modules. Finally, as tests are performed and error data are gathered, fault profiles may be used to rank and classify errors uncovered. The severity of the problem is indicated by the priority. A description of an error is provided by fault categories so that statistical error analysis can be performed.

Q.55. What are the uses of testing tools ? ***(R.G.P.V., June 2016)***

Ans. Software testing tools are used as part of the testing phase within the software development lifecycle to automate certain task and discover issues that might be difficult to find using manual review alone. Software testing tools are frequently used to ensure consistency, thoroughness, and efficiency in testing software products and to fulfill the requirements of planned testing activities. These tools may facilitate unit testing and subsequent integration testing as well as commercial software testing.

Q.56. Discuss software testing tools and its type.

Ans. Software testing tools are often used to make sure consistency, thoroughness, and efficiency in testing software products and to meet the requirements of planned testing activities. These tools may make easy unit or module testing and subsequent integration testing for example, drivers and stubs as well as commercial software testing tools.

Testing tools can be classified in the following manner –

(i) Static Test Tools – Static testing tools help the software engineer in generating test cases. There are three different types of static testing tools used in the industry namely, code-based testing tools, specialized testing languages, and requirements-based testing tools. Code-based testing tools take source code (or PDL) as input and conduct a number of analyses that result in the generation of test cases. Specialized testing languages, for example, ATLAS, are used by software engineers to write detailed test specifications that specify each test case and the logistics for its execution. Requirements-based testing tools isolate specific user requirements and suggest test cases or classes of tests that will exercise the requirements.

(ii) Dynamic Test Tools – Dynamic testing tools deals with an executing program, checking path coverage, testing assertions about the value of certain variables and otherwise instrumenting the execution flow of the program. Dynamic tools are either intrusive or nonintrusive. An intrusive tool alters the software to be tested by inserting probes that conduct the activities just mentioned. Nonintrusive testing tools use a isolated hardware processor that executes in parallel with the processor containing the program that is being tested.

(iii) Test Management Tools – Test management tools are helpful in controling and coordinating software testing for each of the major testing steps. These tools manage and coordinate regression testing, perform comparisons that assess differences between actual and expected output and perform batch testing of programs with interactive human/computer interfaces. Furthermore, many test management tools also work as generic test drivers. A test driver reads one or more test cases from a testing file, formats the test data to conform to the requirements of the software under test and then calls the software to be tested.

(iv) Client/Server Testing Tools – The client/server environment requires, specialized testing tools that exercise the graphical user interface and the network communications needs for client and server.

NUMERICAL PROBLEMS

Prob.1. Explain what is cyclomatic complexity. What are the various methods of calculation of cyclomatic complexity ? Consider a flow graph (fig. 4.21) and calculate cyclomatic complexity by all the three methods.

(R.G.P.V., Dec. 2008)

Sol. **Cyclomatic Complexity and Its Method of Calculation** – Refer to Q.49.

Problem – Now consider the flow graph in fig. 4.21.

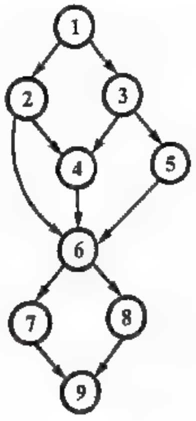


Fig. 4.21

The cyclomatic complexity of the given graph can be calculated on one of the following ways –

(i) The flow graph has 5 regions

(ii) V(G) = 12 edges – 9 nodes + 2 = 5

(iii) V(G) = 4 predicate nodes + 1 = 5

Thus, the cyclomatic complexity of the flow graph in fig. 4.22 is 5.

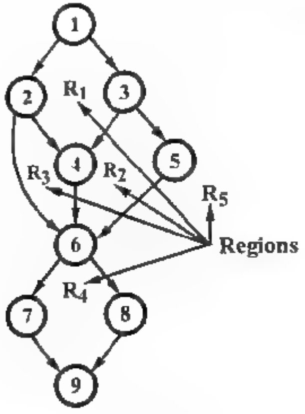


Fig. 4.22

Prob.2. Explain cyclomatic complexity. Compute cyclomatic of a graph (fig. 4.23) shown and also find all independent paths. ***(R.G.P.V., Nov./Dec. 2007)***

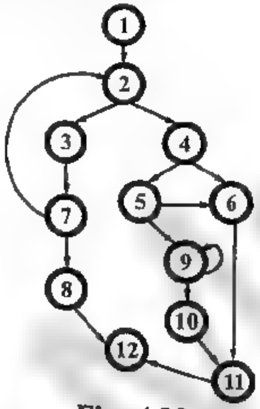


Fig. 4.23

Sol. **Cyclomatic Complexity** – Refer to Q.49.

The cyclomatic complexity of the given graph can be calculated in one of the following ways as –

(i) The flow graph has 6 regions.

(ii) V(G) = 16 edges – 12 nodes + 2 = 6

(iii) V(G) = 5 predicate nodes + 1 = 6

Thus, the cyclomatic complexity is 6.

Independent paths are –

(i) 1 2 – 4 6 – 11 – 12

(ii) 1 – 2 – 4 – 5 – 6 – 11 – 12

(iii) 1 – 2 – 4 – 5 – 9 – 10 – 11 – 12

(iv) 1 – 2 – 4 – 5 – 9 – 9 – 10 – 11 – 12

(v) 1 – 2 – 3 – 7 – 8 – 12

(vi) 1 – 2 – 3 – 7 – 2

Prob.3. Compute the cyclomatic complexity by all three methods for finding greater number between two variables. ***(R.G.P.V., Dec. 2015)***

Sol. The program for finding greater number between two variables is written as –

```
procedure greater;
int x, y, temp = 0;
enter the value of x;
enter the value of y;
if (x > y)
temp = x;
else
temp = y;
return temp;
```

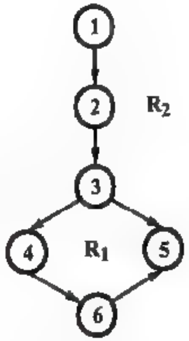


Fig. 4.24

The control flow graph for above program fragment is shown in fig. 4.24. The cyclomatic complexity can be calculated in one of the following ways –

(i) The flow graph has two regions

(ii) V(G) = 1 predicate node +1 = 2

(iii) V(G) = 6 edges – 6 nodes + 2 = 2

Hence, the cyclomatic complexity for the given flow graph is 2.

Prob.4. Calculate the cyclomatic complexity for the following program. Also explain your approach –

```
int temp
if (a > b) temp = a
else temp = b
if (c > temp)
temp = c
return temp
```

(R.G.P.V., June 2011, 2015)

Sol. The given program in C is written as –

```
0    int temp;
1    if (a > b)
2    temp = a;
3    temp = b;
4    if (c > temp)
5    temp = c;
6    return temp;
```

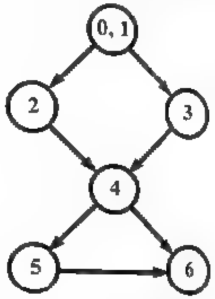


Fig. 4.25 Flow Graph

The control flow graph for this is shown in fig. 4.25.

The cyclomatic complexity can be calculated in one of the following ways –

(i) The flow graph has three regions

(ii) V(G) = 2 predicate nodes + 1 = 3

(iii) V(G) = 7 edges – 6 nodes + 2 = 3

Thus, the cyclomatic complexity is 3.

INTRODUCTION TO OBJECT-ORIENTED ANALYSIS, DESIGN AND COMPARISON WITH STRUCTURED SOFTWARE ENGINEERING

Q.57. Write short note on object-oriented analysis. (R.G.P.V., June 2003)

Ans. Object-oriented analysis (OOA) is the first technical activity that is performed as part of object-oriented software engineering. OOA introduces new concepts for examining a problem. It is grounded in a set of basic principles, which are as follows –

(i) The information domain is modeled.

(ii) Behaviour is represented.

(iii) Function is described.

(iv) Data, functional, and behavioural models are divided to uncover greater detail.

(v) Early models represent the essence of the problem, while later ones provide implementation details.

The above noted principles form the foundation for the OOA approach.

Q.58. Write a short note on object-oriented analysis and design.
(R.G.P.V., Dec. 2004)

Ans. **Object-oriented Analysis** – Refer to Q.57.

Object-oriented Design – The analysis model created using object-oriented analysis is transformed by the object-oriented design into a design model that works as a plan for software construction.

OOD results in a design having several different levels of modularity i.e., major system components are partitioned into subsystems (a system-level "module"), and data and their manipulating operations are encapsulated into objects (a modular form that is the building block of an OO system). Moreover, OOD must specify the certain data organization of attributes and the procedural detail of every single operation.

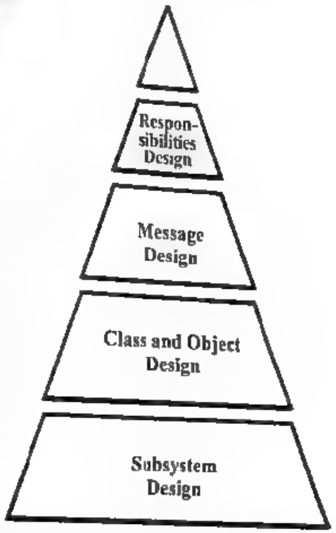


Fig. 4.26 The Object-oriented Design Pyramid

Fig. 4.26 shows a design pyramid for object-oriented systems. It is having the following four layers –

(i) The Subsystem Layer – It represents each of the subsystems that make software enable to get user requirements and to implement the technical infrastructure that helps user requirements.

(ii) The Class and Object Layer – It represents the class hierarchies that make the system enable to be developed using generalizations and specializations. This layer also represents each object.

(iii) The Message Layer – It represents the design details that make each object enable to communicate with its collaborators. It establishes the internal and external interfaces for the system.

(iv) The Responsibilities Layer – It represents the data structure and algorithmic design for all attributes and operations for each object.

The OO design pyramid exclusively emphasizes on the specific product or system design. Note, however, that another design layer is present, which forms the base on which the pyramid rests. This basic layer concentrates on the design of domain objects, which play a key role in constructing the infrastructure for the OO system by providing assistance for human/computer interface activities, task management, and data management.

Q.59. Write in detail about the object-oriented system design and also discuss about UML. ***(R.G.P.V., Dec. 2017)***

Ans. **Object-oriented System Design** – Refer to Q.58.

UML – Refer to Q.18 (Unit-III).

Q.60. What is the purpose of domain analysis ? How is it related to the concept of requirements pattern ? ***(R.G.P.V., June 2008)***

Ans. Object-oriented analysis of systems can happen at many different levels of abstraction. At the business or enterprise level, the techniques related with OOA are coupled with a business process engineering approach in an attemp to define classes, objects, relationships, and behaviours that model the whole business. At the business area level, an object model describing the working of a specific business area can be defined. At an application level, the object model concentrate on certain customer requirements as those requirements affect an application to be constructed.

Domain analysis is carried out when an organization needs to create a library of reusable classes that will be applicable to whole category of applications.

Object-technologies are influenced through reuse. Consider, for an example, the analysis of requirements for a new application specifies that 100 classes are required. Two teams are assigned to construct the application. Each will design and build a final product. Each team has people with the identical skill levels and experience.

Team X does not have access to a class library, and hence, it must develop all 100 classes from scratch.

Team Y uses a robust class library and discovers that 55 classes are already available. It is highly probable that –

(i) Team Y will complete the project much sooner compared to Team X.

(ii) The Team Y's product cost will be extremely lower compared to the Team A's product cost.

(iii) The Team Y's product will have less delivered defects compared to Team A's product.

Q.61. Briefly outline the important steps involved in developing a software system using a popular object-oriented design methodology. ***(R.G.P.V., June 2012)***

Ans. A software engineer should perform the following generic steps to perform object-oriented design –

(i) Describe each subsystem and assign it to processors and tasks.

(ii) Select a design method for implementing data management, interface support, and task management.

(iii) Design a suitable control mechanism for the system.

(iv) Perform object design using a procedural representation for each operation and data structures for class attributes.

(v) Perform message design with the help of colloborations between objects and object relationships.

(vi) Produce the messaging model.

(vii) Review the design model and iterate as needed.

Q.62. Write the benefits of object-oriented design.

Ans. There are following benefits of object-oriented design –

(i) The concept of objects performing services is a more natural way of thinking.

(ii) Objects are inherently reusable.

(iii) Emphasis is on understanding the problem domain.

(iv) Internal consistency of system is enhanced because attributes and services can be considered as an intrinsic whole.

(v) The characteristic of inheritance capitalize on the commonalty of attributes and services.

(vi) The characteristic of information hiding stabilizes systems by localizing changes to objects.

(vii) The object-oriented development process is consistent from analysis, through design, to coding.

Q.63. Differentiate between structured analysis and object-oriented analysis. ***(R.G.P.V., June 2008, 2013, Dec. 2014)***

Or

Compare object-oriented analysis of software with that of structural software engineering approach. ***(R.G.P.V., Dec. 2016)***

Ans. The major differences between structured analysis and object-oriented analysis are given in table 4.3.

Table 4.3 Differences between SA and OOA

S.No.		SA	OOA
(i)	System components	Functions	Objects
(ii)	Data processes, control processes, and data stores	Separated through internal decomposition	Encapsulated within objects
(iii)	Characteristics	· Hierarchical structure · Classification of functions · Encapsulation of knowledge within functions	· Inheritance · Classification of objects · Encapsulation of knowledge within objects

Q.64. Discuss the differences between object-oriented design and function-oriented design strategies. ***(R.G.P.V., Dec. 2004)***

Or

Compare relative advantages of the object-oriented and function-oriented approaches to software design. ***(R.G.P.V., June 2012)***

Ans. There are two common design strategies for software systems - function-oriented and object-oriented. In function oriented, a module is specified by its function. Function-oriented design provides a basis for partitioning in function-oriented approaches i.e., with partitioning of the problem, the overall transformation function for the system is partitioned into several smaller functions comprising the system function. The system decomposition is in terms of functional modules.

Whereas in the object-oriented design strategy any entity in the real world provides some services to the environment to which it belongs. This view is supported by data abstraction. Data is treated not as an object simply, but as objects with some predefined operations on them. The predefined operations on data objects are the only operations performed on those objects. Thus, the basis for object-oriented design is data abstraction.

UNIT 5

SOFTWARE MAINTENANCE AND SOFTWARE PROJECT MEASUREMENT

NEED AND TYPES OF MAINTENANCE, SOFTWARE CONFIGURATION MANAGEMENT (SCM), SOFTWARE CHANGE MANAGEMENT, VERSION CONTROL, CHANGE CONTROL AND REPORTING

Q.1. Write short note on software maintenance.

(R.G.P.V., June 2004, Dec. 2004)

Ans. There are two formal definitions of software maintenance, which are as under –

(i) IEEE, 1993 – Software maintenance is modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

(ii) ISO/IEC Standard 12207 – Software maintenance is a set of activities performed when software undergoes modifications to code and associate documentation due to a problem or the need for improvement or adaptation.

The first definition is similar to hardware maintenance e.g., car servicing, where a product is checked for errors or given additional functions after it has been sold. In contrast, the second definition includes software maintenance as an essential aspect of the entire life cycle of the software product, beginning from its early development.

Software maintenance specifies activities following the delivery of the initial working version of the software system. Maintainability should be built into the system from the beginning i.e., maintenance should be kept in mind throughout development. Although maintenance does not have the appeal of development, it is the most complex aspect of software production. Maintenance includes parts of all the other phases of the software process. After receiving a maintenance request, the first step is to identify what type of maintenance is needed. Occasionally, the problem is with the user – not the software.

In case of software products, one would be more interested to update the product to make it compatible to the changes in its environment or the hardware, instead of eliminating and replacing it with another costlier new product. Hence, software products also require maintenance for keeping them up to date with their surroundings for their best application.

Q.2. Describe the components of software maintenance process. Why cost of software maintenance is high ? (R.G.P.V., Dec. 2010, June 2015)

Ans. In software maintenance process followings are most important –

(i) People, who are involved

(ii) Knowledge (semantics, structural) about the software product

(iii) Supporting tasks, which are well defined.

These three are inter-related and interdependent to each other as shown in fig. 5.1.

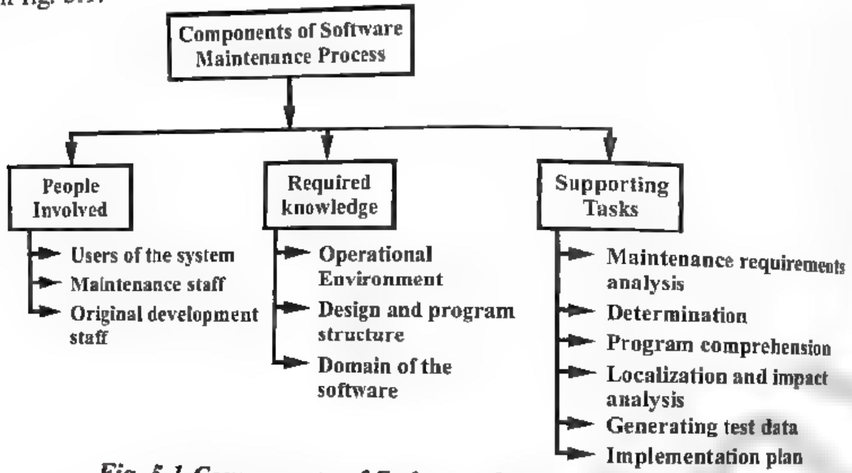


Fig. 5.1 Components of Software Maintenance Process

Software maintenance cost are high because it is more expensive to add functionality after a system is in operation than it is to implement the same functionality during development.

Q.3. Explain the meaning of the term "maintenance" in software engineering and various software maintenance tables. (R.G.P.V., Dec. 2009)

Ans. **Maintenance** – Refer to Q.1.

Software Maintenance Table –

Table 5.1 Software Maintenance Table

Phases	*Input*	*Process*	*Control*	*Output*
Problem identification phase	• Modification request	• Assign change number • Classify modification request • Accept or reject change • Prioritize	• Uniquely identified modification request • Enter modification request repository	• Validated modification request • Validated process determinations
Analysis phase	• Project document • Repository information • Validated modification request	• Feasibility Analysis • Detailed analysis	• Conduct technical review • Verify test strategy • Verify whether the documentation is updated or not • Identify security issues	• Feasibility report • Detailed analysis report • Updated requirements • Preliminary modification list • Test strategy
Design phase	• Project document • Source code • Databases • Analysis phase output	• Create test cases • Revise requirements • Revise implementation plan	• Software inspections/ reviews • Verify design	• Revised modification list • Revised detailed analysis • Updated test plans
Implementation phase	• Source code • System documentation • Results of design phase	• Software code • Unit test • Test preparation review	• Software inspections/ review	• Updated software • Updated design documents • Updated test documents • Updated user documents • Test preparation review report
System test phase	• Updated software documentation • Test preparation review report • Updated system	• Functional test • Interface testing • Test preparation review	• Software code listings • Modification request • Test documentation	• Test system • Test reports
Acceptance test phase	• Test preparation review report • Fully integrated system • Acceptance test plans • Acceptance test cases • Acceptance test procedures	• Acceptance test • Interoperability test	• Acceptance test	• Acceptance test report
Delivery phase	• Tested/accepted system	• Installation • Training	• Version description document	• Version description document

Q.4. What are the different types of maintenance that a software product might need ? Why are these maintenance required ? (R.G.P.V., June 2011, 2014)

Or

What are the various software maintenance activities involved throughout its cycle ? (R.G.P.V., Dec. 2016)

Or

Discuss briefly software maintenance activities. (R.G.P.V., May 2019)

Ans. There are four types of software maintenance, namely, corrective maintenance, adaptive maintenance, perfective maintenance, and preventive maintenance. The distribution of types of maintenance by type and by percentage of time consumed is shown in fig. 5.2.

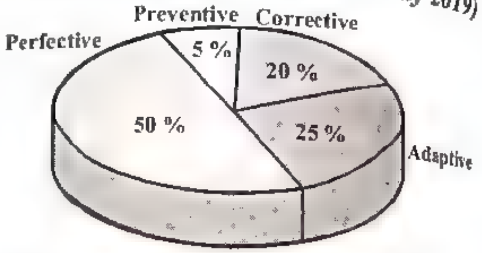


Fig. 5.2 Types of Software Maintenance

(i) Corrective Maintenance – It is concerned with the repair of faults or defects found in daily system functions. A defect can result because of design errors, logic errors, and coding errors. For example, design errors take place when changes made to the software are incorrect, incomplete, wrongly communicated, or the change request is misunderstood. Likewise, logic errors can result due to invalid tests and conclusions, incorrect implementation of design specifications, faulty logic flow or incomplete test of data. All these errors, called as residual errors, prevent the software from conforming to its agreed specification. Corrective maintenance of a software product is necessary either to rectify the bugs observed while the system is in use.

The approach in corrective maintenance is to locate the original specifications in order to determine what the system was originally designed to do. However, because of pressure from management, the maintenance team sometimes resorts to emergency fixes called as 'patching'. Corrective maintenance accounts for 20% of all maintenance activities as shown in fig. 5.2.

(ii) Adaptive Maintenance – It is the implementation of changes in a part of the system, which has been affected by a change that happened in some other part of the system. Adaptive maintenance is composed of adapting software to changes in the environment, like the hardware or the operating system. In this context, the term environment means the conditions and influences which act (from outside) on the system. For example, business rules, work patterns, and government policies have a significant impact on the software system. Adaptive maintenance accounts for 25% of all maintenance activities as shown in fig. 5.2.

(iii) Perfective Maintenance – It is concerned with implementing new or changed user requirements. Perfective maintenance involves making functional enhancements to the system and activities to enhance the system's performance even when the changes have not been proposed by faults. This encompasses enhancing both function and efficiency of the code and changes, insertions, deletions, modifications, extensions, and enhancements made to a system to satisfy the evolving and/or expanding needs of the user.

Modifying the payroll program to incorporate a new union settlement and adding a new report in the sales analysis system are examples of perfective maintenance. Perfective maintenance is the largest user of maintenance activities and accounts for 50% of all maintenance activities as shown in fig. 5.2.

(iv) Preventive Maintenance – It is the maintenance carried out for the purpose of preventing problems before they happen. The aim is to make programs simple to understand. This assists in changing the software to enhance its future maintainability or to give a better base for future enhancements. Preventive maintenance encompasses activities that are aimed at enhancing the system's maintainability, like updating documentation, adding comments, and improving the modular structure of the system. Restructuring and optimizing code are examples of preventive maintenance.

Preventive software maintenance needs an investment of resources in the system. The return on that investment is often difficult for customers to appreciate, but as Lehman's second law says, the more a system is subjected to maintenance, the more its structure will degrade. This results in a system which is more difficult to understand and ultimately more expensive to change. Preventive maintenance accounts for only 5% of all maintenance activities as shown in fig. 5.2.

Q.5. Write short note on software configuration management. (R.G.P.V., Dec. 2005, Nov./Dec. 2007, June 2013)

Or

What is meant by software configuration management ? (R.G.P.V., June 2012)

Or

Brief the software configuration management and its process. (R.G.P.V., Dec. 2017)

Ans. One of the foundations of ***software engineering*** is software configuration management (SCM). SCM is used to track and manage the emerging product and its versions. This is to assure quality of the product during development, and operational maintenance of the product. The design requirements can be traced to the final software product through SCM.

SCM prevents the software process from being unmanageable by controlling change and maintaining system integrity. It recognizes and controls the configuration of software, hardware, and the tools that are used throughout the development cycle.

Different people perceive software configuration management differently. Few perceptions of several people about SCM in the form of definitions are as follows –

(i) SCM is the process of defining and implementing a standard configuration, that results into the primary benefits such as easier setup and maintenance, less down time, better integration with enterprise management, and more efficient and reliable backups.

(ii) Software configuration management (SCM) is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes.

(iii) SCM is the process concerned with the development of procedures and standards for managing an evolving software system product.

(iv) SCM is a set of procedures meant to identify, control, provide, and log the various work products of a software project.

(v) SCM is the ability to control and manage change in a software project.

(vi) In the simplest sense, SCM is the process of controlling baseline software documents and code.

Q.6. Discuss the importance of SCM. Enumerate the goals of SCM. ***(R.G.P.V., Dec. 2010)***

Ans. Software evolution states that, unless steps are taken to maintain control of the software system, the entropy of the system will continually increases. When a system's entropy is increasing, the system becomes more chaotic – It becomes more and more difficult to correct errors and make enhancements.

All organizations exercise some form of software configuration management (SCM), though many would not recognize it as such. It can be a simple as the single programmer that keeps track of the programs progress, errors, and requirements in his head to the large software company that uses strict SCM practices during development.

As programs have become larger and more since the 1960's, the development of SCM techniques has become more important. Developers need SCM to improve productivity, reduce cost and to keep the project progressing to completion. It becomes even more important when software is developed by outsourcing and by distributed development.

SCM gives the developers a means to coordinate the work of numerous people on a common product whether they work in close proximity or over a wide geographical area. Without a good SCM plan, projects become increasingly difficult to control and can reach the point where the project has to be discontinued because it cannot be fixed. There is an impact of software lifecycle model on software configuration management.

Goals of SCM – There are following major goals of software configuration management (SCM) –

(i) Software configuration management activities are planned.

(ii) Changes to identified software work products are controlled.

(iii) Selected software work products are identified, controlled, and available.

(iv) Affected groups and individuals are informed of the status and content of software baselines.

Q.7. What are the objectives and features supported by software configurations management ? ***(R.G.P.V., June 2016)***

Ans. **Objectives of SCM** - Refer to Q.6.

Features of SCM – The basic features provided by any SCM tools are as follows –

(i) Concurrrency Management – When two or more tasks are happening at same time it is known as concurrent operation. In SCM,it means that the same file being edited by multiple persons at the same time. In the absence of concurrency management, SCM tools may lead to very severe problems.

(ii) Version Control – SCM tools uses archiving method or saves every change made to file so that it is possible for use to roll back to previous version in case of any problems.

(iii) Synchronization – SCM tools allow the user to checkout more than one files or entire copyof repository. The user then works on the required files and checks in the changes back to repository, also they can update there local copy of periodically to stay updated with the changes made by other team members.

Q.8. What do you understand by software change management ?

Ans. Software change management is the process of selecting which changes to encourage, which to permit, and which to prevent, according to project criteria like schedule and cost. The process identifies the origin of changes, defines critical project decision points and establishes project roles and responsibilities. Fig. 5.3 shows the necessary pro^c^ess components and their relationships.

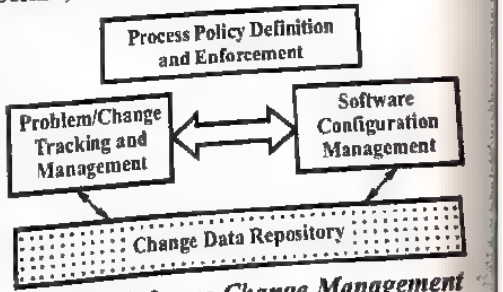

Process Policy Definition and Enforcement
Problem/Change Tracking and Management
Software Configuration Management
Change Data Repository

Fig. 5.3 Software Change Management

Q.9. What are the problems encountered with uncontrolled change management ? ***(R.G.P.V., June 2017)***

Ans. The uncontrolled changes can create very serious problems. These problems are serious enough to sink the whole project and cause project failure. Minor changes can create confusion and chaos. Uncontrolled change can create four most dangerous problems of software development and maintenance which are very common. These problems are multiple maintenance problem, communication breakdown problem, shared data problem and simultaneous update problem.

If the change management is uncontrolled then there will be a chance of getting duplicated efforts on the same problem. If a person is making some changes on any item then because of uncontrolled change management there will be no record of who is making changes and what type of changes, these facts are unknown to other people in the project. So, there will be a chances of occurrence of problem of one person overwriting another person's efforts.

Q.10. What is version control in project ? ***(R.G.P.V., Dec. 2015)***

Ans. The procedures and tools are combined by the version control to manage different versions of configuration items that are created during the software engineering process.

A version of the software is a collection of software configuration items (source code, documents, data). Each version may be consist of different variants. Version control activity is divided in four sub-activities

(i) Identifying New Versions – A software configuration item (SCI) will get a new version number when there has been a change to its established baseline. Each previous version will be stored in a corresponding directory like version 0, version 1, version 2 etc.

(ii) Numbering Scheme – The numbering scheme will have the following format –

Version X.Y.Z .....

The first letter (X) denotes the entire SCI. Therefore, changes made to the entire configuration item, or changes large enough to warrant a completely new release of the item, will cause the first digit to increase.

The second letter (Y) denotes a component of the SCI. This digit will sequentially increase if a change is made to a component or small changes to multiple components.

The third letter (Z) denotes a section of the component of a SCI. This number will only be visible if a component of an SCI can be divided into individual sections. Changes made at this level of detail will need a sequential change of the third digit.

(iii) Visibility – The version number can be seen either in a frame, or below the title. The decision for this relies upon the group decision to code all documents for a frames capable browser or permit for non-frames capable browsers. In either case, the number will always be made available.

(iv) Tracking – The best way to keep track of the different versions is with a version evolution graph as depicted in fig. 5.4.

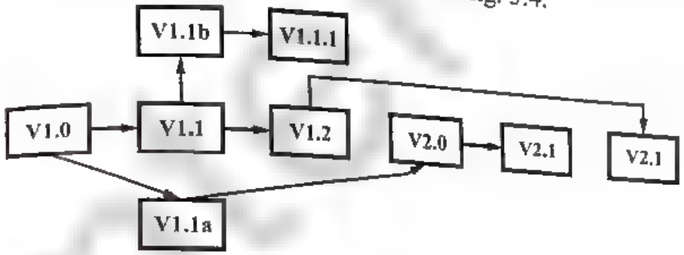


Fig. 5.4 Version Evolution Graph

For example, if we required to keep track of every updated project schedule then we could assign a version number each time a change was made.

Q.11. What is change control ? Explain change control process. ***(R.G.P.V., June 2010)***

Ans. Change control involves procedures and tools to bring order to the change process. Larger projects have a formal change control board (CCB), whose responsibility is to review and approve or disapprove change. It is the CCB responsibility to provide the mechanism to maintain orderly change processes. For a large software engineering project, uncontrolled change rapidly leads to chaos. For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change.

Fig. 5.5 shows the change control process (CCP). Change control procedures make sure that changes to the system are controlled and that their effect on the system can be predicted. A change control form (CCF), or a software problem report (SPR), is filled out to make a change. The form has data on estimation costs, the data when the change was requested, approved, implemented, and validated. The information on the form is defined in the SCM plan and it makes up much of the information in the SCM database. Then this form is provided to the change control board (CCB). Then it is examined by CCB for its validity, the impact of the change and the cost. The change is then approved, or disapproved by the CCB. If it is approved, it is applied to the software and regression testing is performed to be sure that the change has not affected other parts of the system.

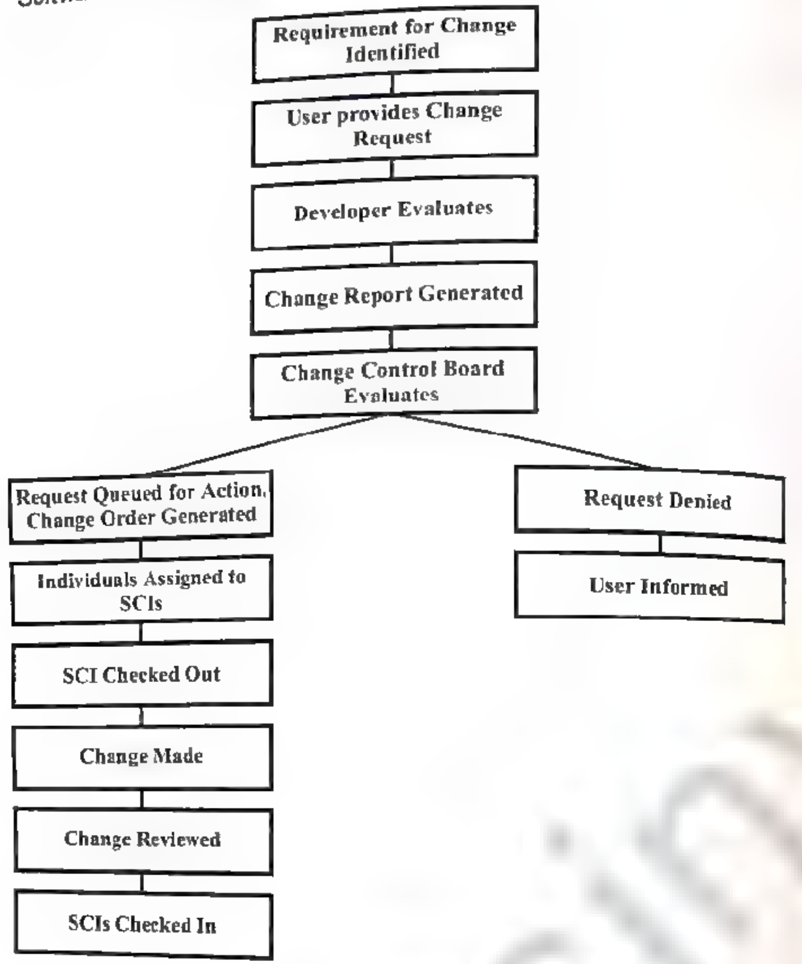

Requirement for Change Identified
User provides Change Request
Developer Evaluates
Change Report Generated
Change Control Board Evaluates
Request Queued for Action, Change Order Generated
Request Denied
User Informed
Individuals Assigned to SCIs
SCI Checked Out
Change Made
Change Reviewed
SCIs Checked In

Fig. 5.5 Change Control Process

Records are kept of all changes approved and disapproved. This permits for history reports to be generated when needed.

Q.12. What is configuration status reporting ?

Ans. Configuration status reporting, also known as status accounting, is an SCM task that records and reports the changes to be incorporated in the work product.

Each time an SCI is allocated new or updated identification, a CSR entry is made. Each time a change is approved by the CCA, a CSR entry is created. Each time a configuration audit is carried out the results are reported as part of the CSR task. Output from CSR may be kept in an on-line database, so that software developers or maintainers can access change information by keyword category. Furthermore, a CSR report is produced on a regular basis and is intended to keep management and practitioners appraised of important changes.

Configuration status reporting plays an important role in the success of a large software development project. When several people are involved, it may happen that the left hand not knowing what the right hand is doing syndrome will take plae. Two developers may try to modify the same SCI with different and conflicting intents. A software engineering team may take months of effort building software to an obsolete hardware specification. The person who would identify severe side effects for a proposed change is not aware that the change is being made. CSR assists to remove these problems by improving communication among all people involved.

PROGRAM COMPREHENSION TECHNIQUES, RE-ENGINEERING, REVERSE ENGINEERING, TOOL SUPPORT

Q.13. What is program comprehension techniques ?

Ans. Program comprehension represents a necessary part of not only software maintenance, but also the whole software process. Program comprehension is used to support for developing complex code and large size code. Program comprehension is inherently difficult for fresher. Program comprehension provides the facility to understand an existing program. Hence program code may be reused for higher-level task. It is required for fresher that learn program comprehension techniques at the beginning stage and also develop the methods to write their own programs so that may easily be comprehended by others program.

Defect discovery strategies is important for real world program maintenance require to be stressed immediately. Hence not easy to see at the time, the ability to comprehend a program coded by someone else will prove to be very useful to programmers working on real world projects in which developers design, test, write code components for respective functionally cohesive areas. When all software components are added to make a functioning system, the source code written by another programmer is difficult to understand. The ability to comprehend existing programs is necessary since programmers switch job frequently and freshers are being added to the projects constantly. Process of organizing program statements into useful groupings is suggested because a major difference between program comprehension abilities of fresher and experience programmers. The largest evident problem is in making the leap from an understanding of individual program code to the tasks being accomplished groups of code or statements. Program functionality is difficult to understand because program functionality is added in an assembly of coordinated program functions, perhaps being included of various program modules.

Fresher will tend to read a line of program code at a time and fail to connective link between the program statements. Fresher can view a program such as they would a cookbook recipe with the expectation that every statement should be executed in order they appear in a program, comprising those codes which are within procedures. Unless they have been experienced to consider a higher-level technique to program comprehension, desirable time could be spent obtaining nowhere. Freshers cannot be taugh effectively how to identify patterns in source statement. Beacons represent common source code fragments which help at the possible presence of high-level tasks. An example of beacon is the swapping operations comprised when two integer are sorted. This type of operation can be obtained with the use of temporary storage locations. Because freshers have trouble recognizing patterns, they do not make good use of beacons. Freshers spent less time viewing important functional areas of the program than experts do. When experts are asked to comprehend as test programs, they employ their general and domain-specific information to develop hypothesis program content, and then find for beacons which signify the availability of corresponding code structure.

Q.14. What do you mean by the term software re-engineering ? Why is it required ? ***(R.G.P.V., June 2012)***

Or

Explain re-engineering. ***(R.G.P.V., Dec. 2010)***

Or

Write short note on re-engineering. ***(R.G.P.V., June 2013)***

Ans. Software re-engineering deals with reimplementing legacy systems to make them more maintainable. Re-engineering may include the redocumenting the system, organizing and restructuring the system, translating the system to a more modern programming language and modifying and updating the structure and values of the system's data. Normally, there is no change in the functionality of the software and so the system architecture also remains the same.

Software re-engineering helps to get information relating to specification, design, and implementation of the existing software system. This information helps in reimplementing the software so that the functionality, performance, or implementation of the existing software system is improved. Software re-engineering also helps in maintaining the functionality of the software and customizes software in such a way that it is easy to add new functions later, when needed.

Q.15. What are the objectives of software re-engineering ?

Ans. There are following objectives of software re-engineering –

(i) Prepare Functional Enhancement – Re-engineering is often used to modify an existing system rather than using it as it is for improving the functionality of the system. Generally, there is difficulty in changing the existing software system due to continuous modifications in it over the years. Thus re-engineering is used as it specifies the characteristics of the existing software system so that they can be compared with the characteristics of target system. In this manner, re-engineering makes it easy to improve the existing system.

(ii) Improve Maintainability – When software system evolve, their maintainability costs increase. Re-engineering is performed to solve this problem, which helps in redesigning the existing system with suitable functional modules and interfaces.

(iii) Enhance Skills of Software Developers to Incorporate Newer Technologies – A number of software systems are being developed at a fast rate with new features and functions. With the change in software systems, software developers tend to migrate to better technologies. The problem worsens when no vendor support is available for software and hardware for older systems. Furthermore, there is the probability of the older system not being compatible with newer systems.

(iv) Improve Reliability – Software reliability can be affected when changes are made to the software systems. This is because when a change is incorporated, it may create additional problems in software, like incompatibility between hardware and software. This problem can be prevented using software re-engineering as it makes sure that changes are incorporated in a proper way without affecting the functionality of the existing software.

(v) Apply Integration – Sometimes, organizations have many stand-alone software systems, which cannot function simultaneously. In such cases, re-engineering is advantageous as it assists organizations by integrating all the systems to work on a common platform.

Q.16. What are the principles of re-engineering ?

Ans. There are following principles of re-engineering –

(i) Abstraction – It produces representation of the target system by emphasizing required characteristics of the existing system and skipping information about the characteristics, which are not needed in target system. The abstraction performed in this manner (upward movement, see fig. 5.6) is called as reverse engineering.

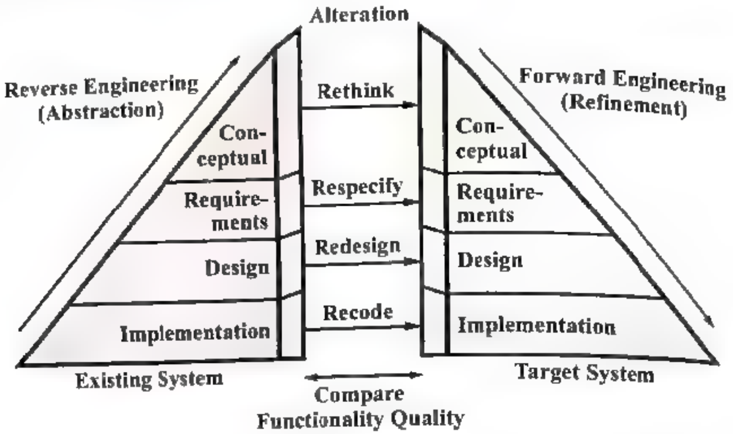


Fig. 5.6 Tasks Performed at Different Levels of Abstraction

(ii) Alteration – It makes one or more changes to the representation of existing system without modifying the extent of abstraction for the target system. The extent of abstraction includes alteration, like addition, deletion, and modification of information. It does not include changing the functionality of system.

(iii) Refinement – It occurs because of successive replacement of information of existing system with more detailed information. Refinement followed in this manner (downward movement, see fig. 5.6) is called as forward engineering.

Q.17. What are the advantages and disadvantages of software re-engineering ?

Ans. There are following advantages of software re-engineering –

(i) Lower Costs – Evidence from a number of US projects suggests that re-engineering makes an existing software system cost significantly less than new system development.

(ii) Lower Risks – Software re-engineering depends on incremental improvement of systems, instead of radical system replacement. The risk of losing critical software knowledge is drastically reduced.

(iii) Better Use of Existing Staff – Existing staff expertise can be maintained, and extended accommodate new skills during software re-engineering. The incremental nature of re-engineering means that existing staff skills can evolve as the system evolves. The approach carries less risk and expense that is associated with hiring new staff.

(iv) Incremental Development – Software re-engineering can be carried out in stages, as budget and resources are available. The operational organization always has a working system and end users are able to gradually adapt to the re-engineered as it is delivered in increments.

The main disadvantage of software re-engineering is that there are practical limits to the extent that a system can be improved by re-engineering. It is not possible, for example, to convert a system written using a functional approach to an object-oriented system. Major architectural changes or radical reorganizing of the system data management cannot be carried out automatically, so involve high additional costs. Although re-engineering can improve maintainability, the re-engineered system will probably not be as maintainable as a new system developed using modern software engineering methods.

Q.18. Explain the software re-engineering process model.

Ans. The software re-engineering process model defines six major activities, which are shown in fig. 5.7. These activities occur in a linear sequence but this sequence is not always the case.

Fig. 5.7 shows that the re-engineering paradigm is a cyclical model. It means that each of the activities of the paradigm may be revisited. The process can terminate after any one of these activities for any particular cycle.

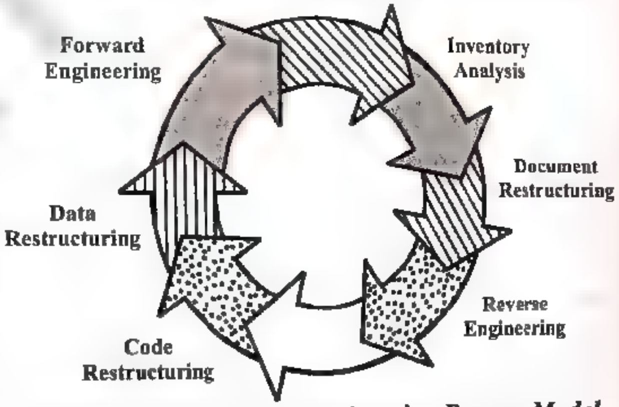


Fig. 5.7 A Software Re-engineering Process Model

(i) Inventory Analysis – Each software organization should possess an inventory of all applications. The inventory is a spreadsheet model that contains information and provides a detailed description of every active application. Candidates for re-engineering appear by sorting this information according to business criticality, longevity, current maintainability, and other locally important criteria. For re-engineering work, resources can be assigned to candidate applications.

It is to be noted on a regular cycle that the inventory should be revisited. The status of applications can change as a function of time and as a result priorities for re-engineering will shift.

(ii) Document Restructuring – The trademark of many legacy systems is poor documentation. Some options are viable for this, which are as follows –

(a) *Creating documentation is so time consuming*. This is the correct approach in some cases. For hundreds of computer programs, it is not possible to recreate documentation. If a program is relatively static, is coming to the end of its useful life, and is unlikely to undergo important change.

(b) *Documentation must be updated, but there are limited resources.* We will use an approach "document when touched". To fully redocument an application, it may not be necessary. Rather than portions of the system which are currently undergoing change are fully documented. A collection of useful and relevant documentation will evolve over time.

(c) *The system is business critical and must be fully documented.* Here, an intelligent approach is to pare documentation to an essential minimum.

(iii) Reverse Engineering – The reverse engineering term contains its origins in the hardware world. To understand the competitor's design and manufacturing secrets, a company disassembles a competitive hardware product in effort. If the competitor's design and manufacturing specifications were obtained then these secrets could be easily understood. But these documents are proprietary and unavailable to the company doing the reverse engineering. For a product, successful reverse engineering introduces one or more design and manufacturing specifications, by examining actual specimens of the product.

For software, reverse engineering is quite similar. However, in most cases, the program to be reverse engineered is not a competitor's. It means that it is the company's own work. The process of analyzing a program is an effort to create a representation of the program at a higher level of abstraction than source code, is the reverse engineering for the software. The reverse engineering is a process of the design recovery. From an existing system, the reverse engineering tools exact data, architectural, and procedural design information.

(iv) Code Restructuring – The code restructuring is the most common type of re-engineering. Some legacy systems have a relatively solid program architecture, but the individual modules were coded, which make them difficult to understand, test, and maintain. The code within the suspect modules can be restructured in such cases.

The source code is analyzed using a restructuring tool, to accomplish this activity. Violations of structured programming constructs are noted and code is then restructured. To ensure that no anomalies have been introduced, the resultant restructured code is reviewed and tested. Internal code documentation is updated.

(v) Data Restructuring – A weak data architectural program will be hard to adapt and improve. So, for many applications, data architecture has more to do with the long-term viability of a program that the source code itself.

The data structuring is a full-scale re-engineering activity, unlike the code restructuring which occurs at a relatively low level of abstraction. Data restructuring starts with a reverse engineering activity in most of the cases. Current data architecture is dissected and necessary data models are defined. For quality, data objects and attributes are identified, and existing data structures are reviewed.

The data are re-engineered, when the data structure is weak. Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.

(vi) Forward Engineering – An application would be reconstructed with the help of an automated "re-engineering engine", in an ideal world. The old program would work as input into the engine, analyzed, restructured and then regenerated in a form which depicted the best aspects of the software quality. It is unlikely that such an engine will appear in the short term but CASE vendors have introduced tools which provide a limited subset of these capabilities which addresses specific application domains. These re-engineering tools are becoming increasingly more sophisticated and important.

The forward engineering recovers design information from existing software and uses this information to change or reconstitute the existing system in an effort to improve its overall quality. Mostly, re-engineered software reimplements the function of the existing system in most of the cases and also adds new functions and/or improves overall performance. The forward engineering is also called as ***renovation*** or ***reclamation.***

Q.19. Discuss the approaches of re-engineering.

Ans. Generally, there are three approaches followed for software re-engineering.

(i) Big Bang Approach – This approach replaces the whole software system at once, that is, it completely replaces the existing system with the target system as shown in fig. 5.8. Because of this, big bang approach is also known as the ***lump sum approach.*** Generally, this approach is used in the projects that require solution to an immediate problem, like migration to different system architecture.

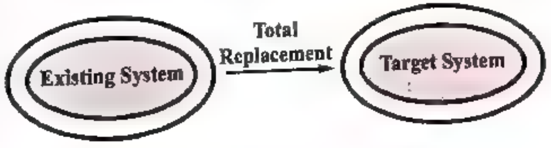


Fig. 5.8 Big Bang Approach

There is no need to develop interfaces between components of existing system and the target system because the big bang approach allows the existing system to be converted into target system all at once. This approach is not always appropriate for large systems because it takes huge amounts of organizational resources and time. The target system needs that the existing software system is in working condition so that the functionalities needed in the target system are easily determined. That is, both the existing system and target system work parallel to each other to make sure improved functionality in the target system.

(ii) Incremental Approach – The incremental approach, also called as ***phase-out approach*** focuses on re-engineering the sections of the software system without considering the functionality of these sections. These sections are added incrementally, as new versions are released to meet user requirements and business objectives. Several versions of the software are released and with the help of these versions, users are view the status of the developing target system.

As shown in fig. 5.9 each component of the existing system is re-engineered separately, that is, their individual re-engineering does not affect the re-engineering of the other components. In the incremental approach the risks in the specific section of software code are identified clearly to help monitor the code. Thus, the incremental approach has a lower risk as compared to the big bang approach.

The benefit of the incremental approach is that it takes less amount of time to produce components of the software system. Another benefit of incremental approach is that the change in existing system is easy to manage.

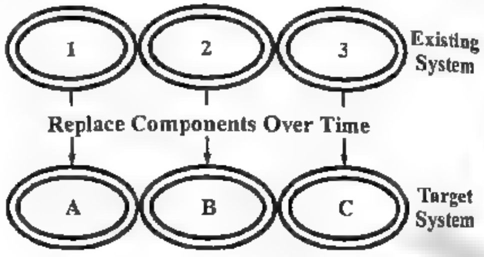


Fig. 5.9 Incremental Approach

(iii) Evolutionary Approach – This approach replaces the sections of the existing system with newly re-engineered sections. However, the sections of existing system are chosen according to their functionality. The target system is developed using the functionally cohesive sections, which are used to determine relationship among the various sections for carrying out re-engineering. Fig. 5.10 depicts the components of existing system, which are split into functions and then re-engineered into new components of the target system.

The benefit of the evolutionary approach is that it results in a modular design. This approach is appropriate when converting existing system to object-oriented systems. Its disadvantage is that similar functions must be identified throughout the existing system and then refined as a single functional unit. Also, interface problems may arise if this approach is used.

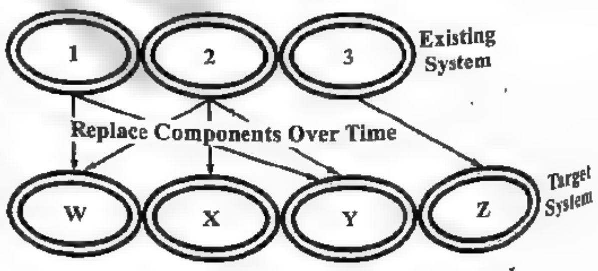


Fig. 5.10 Evolutionary Approach

Q.20. Write short note on reverse engineering.

(R.G.P.V., Nov./Dec. 2007, June 2012)

Or

Explain reverse engineering. ***(R.G.P.V., Dec. 2010, June 2016)***

Ans. The process of analysing software with the objective of recovering its design and specification is reverse engineerng. By the reverse engineering process, the program itself is unchanged. As the input, the software source code is usually available to the reverse engineering process. However, even this has been lost and the reverse engineering process must begin with the executable code.

Both reverse engineering and re-engineering are the different thing. The main objective of reverse engineering is to derive the design or specification of a system from its source code, while the main objective of re-engineering is to produce a new, more maintainable system. The reverse engineering to develop a better understanding of a system is often part of the re-engineering process.

Q.21. How reverse engineering is different from re-engineering process ? ***(R.G.P.V., Dec. 2016)***

Ans. Refer to Q.20 and Q.14.

Q.22. What do you mean by the term software reverse engineering ? Why is it required ? Explain the different activities undertaken during reverse engineering. ***(R.G.P.V., June 2011, 2014)***

Or

Define the term reverse engineering. Explain the different activities undertaken during reverse engineering. ***(R.G.P.V., June 2013)***

Ans. **Reverse Engineering** – Refer to Q.20.

Activities Involved in Reverse Engineering – Fig. 5.11 shows the reverse engineering process. The process begins with an analysis phase. The system is analysed using automated tools to discover its structure during this phase. This is not enough to recreate the system design in itself. Then engineers work with the system source code and its structural model. They add information to this which they have gathered by understanding the system. As a directed graph, this information is maintained which is linked to the program source code.

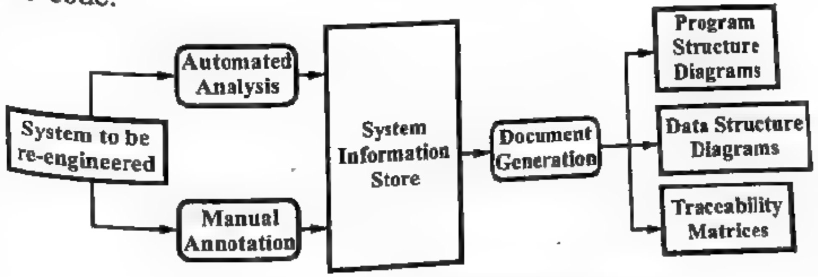


Fig. 5.11 The Reverse Engineering Process

To compare the graph structure and the code and to annotate the graph with extra information, the information store browsers are used. From the directed graph, the various types of documents like program and data structure diagrams and traceability matrix can be generated. The traceability matrices specify where entities in the system are defined and referenced. As the design information is used to further refine the information contained in the system repository, the process of document generation is an iterative one.

To support the reverse engineering process, the tools for program understanding may be used. Usually, these present different system views and permit easy navigation through source code.

Further information may be added to the information store, after the system design documentation has been generated, to help recreate the system specification. Usually, this includes further manual annotation of the system structure. From the system model, the specification cannot be deduced automatically.

Q.23. What are the advantages of reverse engineering process ?

(R.G.P.V., June 2015)

Ans. The following are the advantages related with reverse engineering–

(i) It concentrates on recovering the lost information from the programs.

(ii) It provides the abstract information from the detailed source code implementation.

(iii) It improves system documentation that is either incomplete or out of date.

(iv) It detects the adverse effects of modification in the software system.

(v) It manages the complexity that is present in the software programs.

Q.24. Discuss the types of reverse engineering.

Ans. There are following three important types of reverse engineering –

(i) Redocumentation – One of the oldest forms of reverse engineering is the redocumentation. Redocumentation is one way of producing accurate documentation for an existing software system.

The lack of detailed, accurate and up-to-date program documentation is critical for software engineers and technical managers responsible for the evolution of existing software systems. Without it, the only reliable and objective information is the source code itself. Personnel must spend an inordinate amount of time trying to create an abstract representation of the system's high-level functionality by exploring its low-level source code.

Redocumentation is the process of retroactively providing documentation for an existing software system. If the redocumentation takes the form of modifying commentary within source code, it can be considered a weak form of restructuring.

(ii) Structural Redocumentation – An understanding of the structural aspects of the system's architecture is more important than any single algorithmic component for large legacy systems.

Program understanding is difficult for software engineers and technical managers responsible for the maintenance of such systems. The documentation available for these systems usually describes separate parts of the system. It does not describe the overall architecture.

The documentation is often scattered throughout the system and on different media. It is left to maintenance personnel to explore the low-level source code and piece together disparate information to form high-level structural models. Manually creating just one such architectural document is always difficult, creating the necessary documents that describe the architecture from multiple points of view is often impossible. Yet it is exactly this type of in-the-large documentation that is required to expose the structure of large software systems.

Using reverse engineering to reconstruct the architectural aspects of software may be termed structural redocumentation as proposed by researchers in 1995. As with redocumentation, structural redocumentation does not involve physically restructuring the code.

(iii) Design Recovery – Design recovery is a sub-area of reverse engineering that uses domain knowledge, external and/or informal information, and heuristics, in addition to traditional source-level analyses, to aid program understanding. Its goal is to reproduce all the information required for someone to fully understand the subject system.

Teleological maintenance is related to design recovery, which is an attempt to recover information from the subject system on the basis of a specific user model, for example business rules, rather than from the source code.

Q.25. Describe tool support.

Ans. Software development organizations application tools to help several of the activities which are required to convert a set of user needs into a desired product. A result of computer-aided software engineering (CASE) tools are used for automate the analysis, implementation, design and maintenance of software products. The task is to select application tools wisely to help the technical needs of the product developers and the organization business aims.

In a typical case, several tools are employed over the project life. The degree to which they interoperate and help the process of development may have a large effect on the development team productivity and the quality of the resultant products.

PROJECT MANAGEMENT CONCEPTS, FEASIBILITY ANALYSIS, PROJECT AND PROCESS PLANNING, RESOURCE ALLOCATIONS

Q.26. Explain project management concept. ***(R.G.P.V., Dec. 2009)***

Ans. The successful project management focuses on four P's – people, product, process, and project. The order is not arbitrary. The manager who does not remember that software engineering is an intensely human endeavour will never have successful project management. A manager who does not establish comprehensive interface with the customer at the evolution of project risks building, wrong project. The manager who ignores process, takes risk of inserting competent technical methods and tools into a vacuum. The manager who starts without a solid project plan will sacrifice the success of the product.

The People – The people factor plays a vital role in software engineering. Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

The Product – Before planning the project, the product objectives and scope should be clear. The product objectives identify the overall goals for the product without considering how these goals will be met. Scope identifies the primary data, functions, and behaviours that characterize the product and ways to bound these characteristics in a quantitative manner.

After identifying objectives and scope of the product, alternative solutions are suggested to enable managers to select a best approach.

The Process – It provides framework for planning software development tasks. A number of different task sets – tasks, milestones, work products, and quality assurance enable the framework activities to be adapted to the characteristics of the software project and requirements of the project team.

The Project – So as to manage a successful software project, we must understand what can go wrong and how to do it right.

Q.27. Explain activities covered by the software project management. ***(R.G.P.V., May 2018)***

Ans. The various project management activities are as follows –

(i) Proposal Writing – It is the first step in a software project. The proposal describes the project objectives and the way it will be carried out. Usually, it includes cost and schedule estimates.

Proposal writing is a critical task, since the existence of several software firms depends on having sufficient proposals accepted and contracts awarded. It is a skill which is gained by the experience, there can be no set guidelines for the task of proposal writing.

(ii) Project Planning and Scheduling – Project planning is concerned with identifying the activities, milestones, and deliverables produced by a project. A plan must then be built up to lead the development towards the project goals.

(iii) Project Costing – Cost estimation is an activity related with planning and scheduling. It is concerned with estimating the required resources for the accomplishment of the project plan.

(iv) Project Monitoring and Reviews – Project monitoring is an ongoing project activity. The manager must keep an eye on the project's progress and compare actual and planned progress and costs.

During a project, there must be a number of formal, project management reviews. They are concerned with reviewing overall progress and technical development of the project.

(v) Personnel Selection and Evaluation – Usually, project managers have to select people to work on their project, and ideally, skilled staff having sufficient experience will be available to work on the project. But, in some cases, managers have to settle with a less than ideal project team. However, problems are likely, unless at least one project member has some experience of the type of system being developed. Lack of this experience may cause many simple mistakes.

(vi) Report Writing and Presentation – The project manager is usually responsible for reporting on the project to both the customer and contractor firms. He must write concise, coherent documents which abstract critical information from detailed project reports. He must also be able to present this information while progress reviews are conducted. Consequently, the project manager must essentially be skilled with the ability to communicate effectively both orally and in writing.

Q.28. What is feasibility analysis ? ***(R.G.P.V., June 2016)***

Ans. The process of software development begins with a requirement of software. The reqirement problem first needs to be analysed for its expected budget, durability, time, usability, reliability and compatibility with the existing environment and its necessity in the scenario. The practicality of development, installation, and use of the system is analysed and evaluated in real time after looking at all possible constraints. Fig. 5.12 shows the feasibility analysis activity.

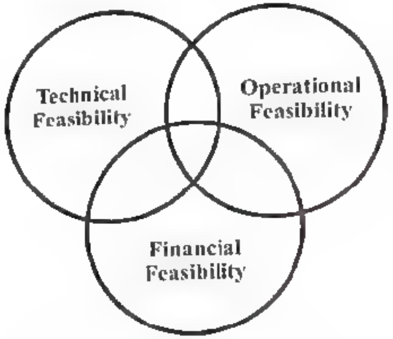


Fig. 5.12

Q.29. What do you mean by feasibility study ? Write its objectives.

Ans. Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software or not. Such information as resource availability, cost estimates for software development, benefits of the software to organization after it is developed, and cost to be incurred on its maintenance are considered.

Objectives – The objectives of feasibility study are –

(i) To establish the reasons for developing a software that is acceptable to users, adaptable to change and conformable to established standards.

(ii) To analyze whether the software will meet organizational requirements or not.

(iii) To determine whether the software can be implemented using the current technology and within the specified budget and schedule or not.

(iv) To determine whether the software can be integrated with other existing software or not.

Q.30. How many types of feasibility analysis apply in a project ?
(R.G.P.V., Dec. 2015)

Ans. There are following types of feasibility –

(i) Technical Feasibility – Technical feasibility assesses the current resources (such as software and hardware) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. Technical feasibility performs the following tasks –

(a) It analyzes the technical skills and capabilities of the software development team members.

(b) It determines whether the relevant technology is stable and established or not.

(c) It ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

(ii) Operational Feasibility – Operational feasibility assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources and involves visualizing whether or not the software will operate after it is developed and be operative once it is installed.

Operational feasibility performs the following tasks –

(a) It determines whether the problems anticipated in user requirements are of high priority or not.

(b) It determines whether the solution suggested by the software development team is acceptable or not.

(c) It analyzes whether users will adapt to a new software or not.

(d) It determines whether the organization is satisfied by the alternative solutions proposed by the software development team or not.

(iii) Economic Feasibility – Economic feasibility determines whether the required software is capable of generating financial gains for an organization or not. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study etc.

A software is said to be economically feasible if it focuses on the following issues –

(a) Cost incurred on software development to produce long-term gains for an organization.

(b) Cost required to conduct full software investigation (such as requirements elicitation and requirements analysis).

(c) Cost of hardware, software, development team, and training.

Q.31. Write short note on project planning.
(R.G.P.V., Nov./Dec. 2007, June 2013)

Ans. Effective management of a software project depends on thoroughly planning the progress of the project. The project manager must anticipate problems which might arise and prepare tentative solutions to those problems. A plan, drawn up at the start of a project, should be used as the driver for the project. This initial plan should be the best possible plan given the available information. It evolves as the project progresses and better information becomes available.

The pseudocode shown in fig. 5.13 describes the project planning process for software development. It shows that planning is an iterative process which is only complete when the project itself is complete. As project information becomes available during the project, the plan must be regularly revised. The overall goals of the business are an important factor which must be considered when formulating the project plan. As these change, changes to the project plan are necessary.

The planning process starts with an assessment of the constraints (required delivery rate, staff available, overall budget, etc.) affecting the project. This is carried out ın conjunction with an estimation of project parameters such as its structure, size, and distribution of functions. The progress milestones and deliverables are then defined. The process then enters a loop. A schedule for the project is drawn up and the activities defined in the schedule are initiated or given permission to continue. After sometimes (usually about 2-3 weeks), progress is reviewed and discrepancies noted. Because initial estimates of project parameters are tentative, the plan will always need to be modified.

```
Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
while project has not been completed or cancelled loop
    Draw up project schedule
    Initiate activities according to schedule
    Wait (for a while)
    Review project progress
    Revise estimates of project parameters
    Update the project schedule
    Renegotiate project constraints and deliverables
    if (problems arise) then
        Initiate technical review and possible revision
    end if
end loop
```

Fig. 5.13 Project Planning

Project managers revise the assumptions about the project as more information becomes available. They replan the project schedule. If the project is delayed, they may have to renegotiate the project constraints and deliverables with the customer. If this renegotiation is unsuccessful and the schedule cannot be met, a project technical review may be held. The objective of this review is to find some alternative approach to development which falls within the project constraints and meets the schedule.

Of course, wise project managers do not assume that all will go well. Problems of some description nearly always arise during a project. The initial assumptions and scheduling should be pessimistic rather than optimistic. There should be sufficient contingency built into the plan that the project constraints and milestones need not be renegotiated every time round the planning loop.

Q.32. Discuss the tools and techniques used in project management.
(R.G.P.V., Dec. 2017)

Ans. The importance of business objectives is to select the right project. The organization support for the project management and having common selection criteria for all initiatives to select the project.

Some of the techniques used in project management are as follows –

(i) Feasibility Study – The term feasibility study is associated with a test of a system proposal according to its workability, impact on the organization, ability to meet user needs, and effective use of resources. It is focuses on three major questions –

(a) The demonstrable needs of the user's and how does the candidate system meet them ?

(b) The resources which are available for a given candidate systems ? Is the problem worth solving ?

(c) What is master MIS plan and how does it fit with the organization and what are the likely impact of the candidate system on the organization ?

(ii) Payback Period – The payback method (PB) is the traditional method of capital budgeting. It is the simplest and, perhaps, the most widely employed quantitative method for appraising capital expenditure decisions. This method helps to find how many years will it take for the cash benefits to pay the original cost of an investment, normally disregarding salvage value ? The payback method (PB) measures the number of years required for the CFAT to payback the original outlay required in an investment proposal.

There are two ways of calculating the PB period. The first method can be applied when the cash flow stream is in the nature of annuity for each year of the project's life, that is, CFAT are uniform. In such a situation, the initial cost of the investment is divided by the constant annual cash flow i.e.,

$$\text{PB} = \frac{\text{Investment}}{\text{Constant annual cash flow}}$$

The second method is used when a project's cash flows are not uniform (mixed stream) but vary from year to year. In such a situation, PB is calculated by the process of cumulating cash flows till the time when cumulative cash flows become equal to the original investment outlay.

(iii) NPV Method – Net present value (NPV) is found by subtracting a projects initial investment from the present value of its cash inflows discounted at the firm's cost of capital.

NPV may be described as the summation of the present values of cash proceeds (CFAT) in each year minus the summation of present values of the net cash outflows in each year. Symbolically, the NPV for projects having conventional cash flows would be –

$$\text{NPV} = \sum_{t=1}^{n} \frac{CF_t}{(1+K)^t} + \frac{S_n + W_n}{(1+K)^n} - CO_0$$

If cash outflow is also expected to occur at some time other than at initial investment (non-conventional cash flows) the formula would be -

$$\text{NPV} = \sum_{t=1}^{n} \frac{CF_t}{(1+K)^t} + \frac{S_n + W_n}{(1+K)^n} - \sum_{t=0}^{n} \frac{CO_t}{(1+K)^t}$$

The decision rule for a project under NPV is to accept the project if the NPV is positive and reject if it is negative. Symbolically,

(a) NPV > zero, accept, (b) NPV < zero, reject.

Zero NPV implies that the firm is indifferent to accepting or rejecting the project. However, in practice it is rare if ever such a project will be accepted, as such a situation simply implies that only the original investment has been recovered.

Q.33. What are the objectives and activities of project planning ? ***(R.G.P.V., June 2005)***

Ans. **Objectives** – The basic goal of planning is to look into the future, identify the activities that need to be done to complete the project successfully, and plan the scheduling and resource allocation for these activities.

The other objectives of project planning are –

(i) It defines the roles and responsibilities of the project management team members.

(ii) It ensures that the project management team works according to business objectives.

(iii) It checks feasibility of schedule and user requirements.

(iv) It determines project constraints.

Activities of Project Planning – The input to the planning activity is the requirements specification and the output is the project plan. i.e., planning activities are –

(i) Cost estimation (ii) Schedule and milestones
(iii) Staffing personnel plan (iv) Software quality assurance plans
(v) Configuration management plans
(vi) Project monitoring plans
(vii) Risk management.

Q.34. What activities are included in project planning ? Where does project planning fit in SDLC ? ***(R.G.P.V., June 2006)***

Ans. **Activities in Project Planning** – Refer to Q.33.

Project Planning in SDLC – Project initiation and planning is the second phase of SDLC. In this phase there are two major activities which are formal investigation of the system problem and the presentation of reasons why the system should or should not be developed by the organization. A critical step at this point is determining the scope of the proposed system. The project leader and initial team of system analysis also produce a specific plan, which the team will follow along with the SDLC step. This baseline project plan customizes the standardize the SDLC and specifies the time and resources needed for its execution.

The project leader and the other with the job of deciding which projects the organization will undertake usually make the final presentation of the business case for proceeding with the subsequent project phases.

Finally, we can say that this phase of SDLC recognizes potential of information system project and gives a detailed plan for conducting the remaining phase of the system development life cycle, for the proposed system. Potential of information system gives a detailed for conducting the remaining phases of the system development life cycle for the proposed system.

Q.35. Explain the various SDLC activities as outlined by ISO 12207 with a neat diagram. ***(R.G.P.V., May 2018)***

Ans. ISO 12207 establishes a common frame-works for software life cycle processes, with well-defined terminology, that can be referenced through software industry. It contains the processes, activities, and tasks that are to be applied during the acquisition of a software product or service and during the supply, development, operation, maintenance and disposal of software products. The software includes the software portion of firmware.

ISO 12207 applies to the acquisition of systems and software products and services. The development of any system use to supply the operation, maintenance and disposal of software products and the software portion of a system, whether performed internally or externally to an organization.

ISO 12207 also provides a process that can be employed for defining, controlling and improving software life cycle processes. The software engineering containing the process of SDLC which is an international standard processes. It was first introduced in 1995 and it aims to be a primary standard that defines all the processes required for developing and maintaining software systems, including the outcomes or activities of each process.

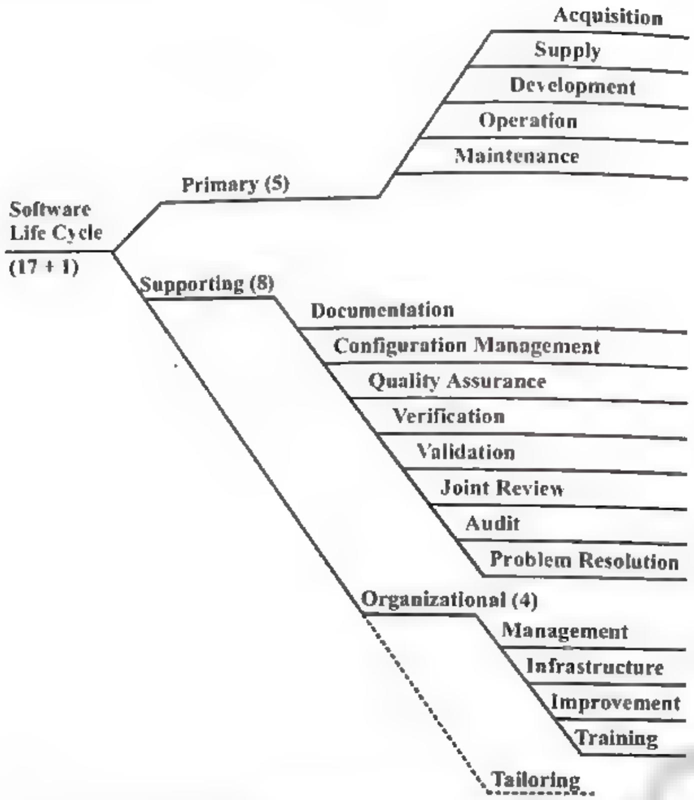


Fig. 5.14 ISO 12207 Life Cycle Process Framework

Q.36. Explain and differentiate between deliverable, baseline and milestones. ***(R.G.P.V., Dec. 2009)***

Ans. **Deliverable** – A deliverable is a project result that is delivered to the customer. It is usually delivered at the end of some major project phase such as specification, design, etc.

Deliverables are usually milestones but milestones need not to be deliverable milestones may be internal project result that are used by the project manager to check project progress but which are not delivered to the customer.

Baseline – When a change is required, a basis for the change should be clearly established prior to making changes to the configuration items. This requires a clear understanding of the configuration items to avoid undesirable results. As a result a ***baseline*** is established when the product is said to be in a stable state. A baseline helps to control change without seriously effecting the justifiable change.

A baseline is referred to as a collection of software configuration items, which have been approved after carrying out formal technical reviews. For example, in the design model errors have been detected and corrected after carrying out the review process. After all the parts of this model have been reviewed, corrected and then approved, the design model can be referred to as a baseline.

Milestones – Milestones are formal representations of the progress of a project. Generally milestones are planned when deliverables are provided. Milestones describe the end-point when software process activity is completed. After completion of a milestone, its output is described in the form of document. This document comprises information about the completion of a phase of the project.

It is difficult to keep in mind all the task(s) being performed during software development. Hence, each task is recorded in documents, which describe the work being done in that phase, with the help of these documents, it becomes easy for the project manager to check the status of the project.

Q.37. What is SCM ? Explain the concept of baseline and SCM items in brief. ***(R.G.P.V., May 2018)***

Ans. SCM – Refer to Q.5.

Baseline – Refer to Q.36.

SCM Items – The software configuration item are the items or aggregation of hardware or software or both that is designed to managed as a single entity. it is an software entity that has been established as a configuration item.The software configuration management controls a variety of items including the code itself. The software items with potential to become SCIs including plans, testing materials, software tools, specifications and design documentation, source and executable code, code libraries, data and data dictionaries, and documentation for installation, maintenance, operations, and software use. It is an important process to select SCI's in which a balance must be achieved between providing adequate visibility for roject control purposes and providing a manageable number of controlled items.

Q.38. Describe resource allocation in project management.

Ans. In the most effective and economical methods, the process of allocating and scheduling is known as resource allocation. Resource allocation is the scheduling of activities for project management. For strategic planning, resource allocation represents a plan for using available resources (i.e., human resources), particularly in the near term, to obtain goals for the future.

Project will always require resources and resources are scarce. The following techniques are used to solve resource allocation problems –

(i) A manual technique (ii) An algorithmic technique
(iii) A combination of both.

Resource allocation is decided with computer programs applied to a specific domain to automatically and dynamically distribute resources of applicants.

SOFTWARE EFFORTS, SCHEDULE AND COST ESTIMATIONS, PROJECT SCHEDULING AND TRACKING, RISK ASSESSMENT AND MITIGATION, SOFTWARE QUALITY ASSURANCE (SQA), PROJECT PLAN, PROJECT METRICS

Q.39. Write a short note on software efforts, schedule and cost estimations.

Ans. **Software Effort** – Effort is required for developing the software product of an estimated size.

Schedule – The duration is required for developing the defined amount of software.

Cost – Expenses are required for developing software project of defined size, comprising the expense of developer effort and labour.

Q.40. What are the approaches to software project estimation ?

Ans. There are three approaches to software project estimation –

(i) Decomposition Technique – It takes a 'divide and conquer' approach to software project estimation.

(ii) Empirical Estimation Model – It can be used as complement for decomposition techniques and offers a potentially valuable estimation approach.

(iii) Automated Estimation Tool – One or more decomposition techniques together with empirical models when combined with graphical user interface generates automated estimation tools which provide an attractive option for estimating.

Q.41. Explain decomposition techniques. Differentiate between LOC based estimation and FP based estimation. ***(R.G.P.V., Dec. 2005)***

Ans. Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major functions and related software engineering activities cost and effort estimation can be performed in a stepwise fashion.

Software cost estimation is a form of problem solving, and in most cases to problem to be solved is too complex to be considered in a single form. Therefore, the problem is decomposed into components in order to achieve an accurate cost estimate. Two approaches are mainly used for decomposition – ***problem-based estimation*** and ***process-based estimation***.

For LOC based estimation and FP based estimation refer to Q.43.

Also, refer Q.42 for process-based estimation.

Q.42. Discuss process-based estimation. Explain it with the help of an example.

Ans. ***Process-based estimation*** is the most common technique of project estimation, in which, estimates are based on the process that will be used. In other words, the process is decomposed into a relatively small set of tasks and the effort required to finish each task is estimated.

Process-based estimation starts with a delineation of software functions obtained from the project scope. For each function, a series of software process activities must be performed. Table 5.2 represents some activities of which functional and some related software process activities may be a part.

Table 5.2 Melding the Problem and the Process

Common Process Framework Activities	*Customer Communication*	*Planning*	*Risk Analysis*	*Engineering*	
Software Engineering Tasks					
Product Functions					
Text Input					
Editing and Formatting					
Automatic Copy Edit					
Page Layout Capability					
Automatic Indexing and TOC					
File Management					
Document Production					

Once melding the problem functions and process activities, the planner estimates the effort that will be required to accomplish each software process activity for each software function. These data is having the central matrix of the table 5.2. Then, average rates of labours are applied to the effort estimated for each process activity. It may be possible that the rate of labours will vary for each task. Senior staff involved in early stages are generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Now as a last step, costs and effort for each function and software process activity are computed.

Illustration of the use of process-based estimation can be given by considering the CAD software. The system configuration and all software functions kept unchanged and are indicated by project scope.

Table 5.3 shows the completed process-based table where estimation of effort for each software engineering activity are provided for each CAD software function.

Table 5.3 Process-based Estimation Table

Activity →	*CC*	*Planning*	*Risk Analysis*	*Engineering*		*Construction Release*		*CE*	*Totals*
Task →				*Analysis*	*Design*	*Code*	*Test*		
Function									
▼									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.75	16.50		45.25
% Effort	1%	1%	1%	8%	45%	10%	36%		

CC= Customer Communication CE = Customer Evaluation

Observe the table 5.3, here the engineering and construction release activities are subdivided into the major software engineering tasks. Also, gross estimates of effort are provided for the customer communication, planning, and risk analysis. These are given in the "total" row at the bottom. Horizontal and vertical totals provide a view of estimated effort required for analysis, design, code, and test. Note that 53% of the overall effort is spent on front-end engineering tasks, i.e., requirements analysis and design, indicating the relative need of this work.

The total estimated project cost (taking an average burdened labor rate of $8000 per month) is $368,000 and the estimated effort is 46 person-month. If needed, labor rates could be associated with each software process activity and computed separately.

Total estimated effort for the CAD software ranges from at least 46 person-months and at most 58 person-months. The average estimate is 53 person-months. The maximum variation from the average estimation is 13% approximately.

Q.43. Discuss LOC and FP based software project estimation.

Or

Explain problem-based estimation techniques.

Ans. During software project estimation, LOC (line of code) and FP (functional point) data are used in the following two ways –

(i) As an estimation variable, to size each element of the software

(ii) As baseline metrics, collected from historical projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC and FP estimation are different and distinct estimation techniques. However, both have some common characteristics. The planner starts with a bounded statement of software scope from which he attempts to decompose software into problem functions that can be estimated separately. LOC and FP is then estimated for each function.

Baseline productivity metrics i.e., LOC/pm or FP/pm are then applied to an appropriate estimation variable, from which, cost or effort for the function is derived. Function estimates are combined to have an overall estimate for the whole project.

However, it should be noted that there is often substantial scatter in productivity metrics for an organization, with the help of a single baseline productivity metric suspect. Generally, LOC/pm or FP/pm averages should be computed by project domain, i.e. projects should be categorized by team size, application area, complexity, and other relevant parameters. After that, local domain averages should be computed. A new project when estimated should first be allocated to a domain, and then the appropriate domain average for productivity should be used in estimation.

The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of partitioning. While using LOC as the estimation variable, decomposition is absolutely necessary and is often taken to considerable levels of detail.

For FP estimates, decomposition works differently. Unlike LOC estimation, each of the information domain characteristics, i.e. inputs, outputs, data files, inquires, and external interfaces are estimated, alongwith the complexity adjustment values. The resultant estimates can then be used to derive a FP value that can be tied to past historical data and used for estimation.

Regardless of the used estimation variable, the project planner starts by estimating various values for each function or information domain value. Using the past data or anticipation, the project planner estimates an optimistic, obvious, and pessimistic size value for each function or count for each information domain value. When a range of values is specified, an implicit indication of the degree of uncertainty is provided.

Then, a three-point or expected value can be computed. The expected value for the estimation variable i.e., size S, can be found as a weighted average of the optimistic (S_{opt}), most likely (S_m), and pessimistic (S_{pess}) estimates. For example –

$$S = (S_{opt} + 4S_m + S_{pess})/6$$

It gives heaviest credence to the "most likely" estimate and follows a beta probability distribution, assuming that there is a negligible probability that the actual result will fall outside the optimistic or pessimistic values.

***Q.44. Explain empirical estimation models. What are its characteristics?* (R.G.P.V., Dec. 2005)**

Or

***Explain empirical estimation models.* (R.G.P.V., June 2008)**

Ans. Empirical estimation model is an estimation model for computer software which uses empirically derived formulas to predict effort as a function of LOC or FP. he empirical data that support most estimation models are derived from a limited sample of projects. Thus no estimation model is suitable for all software classes and in all development environments.

(i) The Structure of Estimation Models – A typical estimation model is derived by having regression analysis on previously collected data. The overall structure of such models are in the following form –

$$E = A + B \times (ev)^C$$

where A, B and C are constants derived empirically, E is effort (person-months), and ev is the estimation variable (may be LOC or FP). The majority of estimation models have some form of project adjustment component with the help of which E can be adjusted by other project characteristics. Among the many LOC-oriented estimation models, proposed in the literature are as follows –

$E = 5.2 \times (KLOC)^{0.91}$	Walston-Felix model
$E = 5.5 + 0.73 \times (KLOC)^{1.16}$	Bailey-Basili model
$E = 3.2 \times (KLOC)^{1.05}$	Boehm simple model
$E = 5.288 \times (KLOC)^{1.047}$	Doty model for KLOC > 9

FP-oriented models have also been proposed. They are as follows –

$E = -13.39 + 0.0545$ FP	Albrecht and Gaffney model
$E = 60.62 \times 7.728 \times 10^{-8}$ FP^3	Kemerer model
$E = 585.7 + 15.12$ FP	Matson, Barnett and Mellichamp model

From the above noted models it is clear that the estimation models must be calibrated for local needs.

(ii) The COCOMO Model – Boehm introduced a hierarchy of software estimation models bearing the name COCOMO (Constructive Cost Model). It has evolved into a more comprehensive estimation model, called COCOMO II.

Like all estimation models for software, the COCOMO II models require sizing information. The three different sizing options available are object points, function points and lines of source code.

The COCOMO II application composition model uses object points. It should be noted that other, more sophisticated estimation models (using FD and KLOC) are also available as part of COCOMO II.

(iii) The Software Equation – The software equation is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project. The model can be derived from data obtained by studying several existing projects. To calculate effort, use the following equation –

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4)$$

where, E = Effort in person-months or person-years
t = Project duration in months or years
B = Special skills factor
P = Productivity parameter.

It indicates the maturity of overall process and management practices. This parameter also indicates the level of programming language used, skills, and experience of software team, and complexity of software application.

Typical values might be P = 2,000 for development of real-time embedded software; P = 10,000 for telecommunication and systems software; P = 28,000 for business systems applications.

***Q.45. Write short note on automated estimation tools.* (R.G.P.V., June 2003)**

Or

***Write short note on automation estimation model.* (R.G.P.V., June 2005)**

Ans. The decomposition techniques and empirical estimation models are available as part of a range of software tools. Such automated estimation tools help the planner to estimate cost and effort and to conduct "what-if" analysis for important project variables, like delivery data or staffing. All the automated estimation tools exhibit the same general characteristics, and all perform the following generic functions –

(i) ***Sizing of Project Deliverables*** – The size of one or more work products is estimated i.e., the external representation of software, the software itself, functionality delivered, descriptive information, are all estimated before.

(ii) ***Selecting Project Activities*** – The required process framework is selected and the software engineering project is specified.

(iii) Predicting Staffing Levels – The number of people available is specified. This is an important function, as the relationship between people available and work is highly nonlinear.

(iv) Predicting Software Effort – Estimation tools use some models relating the project deliverables' size to the required effort (producting them).

(v) ***Predicting Software Cost*** – Software costs can be computed by allocating labor rates to the project activities.

(vi) ***Predicting Software Schedules*** – After having the knowledge of effort, staffing level, and project activities, a draft schedule can be produced by allocating labor across software engineering activities based on recommended models for effort distribution.

Applying different estimation tools to the same project data, a relatively large change in estimated results is found. Also, more importantly, predicted values are often significantly different than actual values. This strengthens the notion that the output of estimation tools should be used as one data point from which estimates are derived.

From these data the model implemented by the automated estimation tool provides estimates of the effort required to complete the projects, costs, staff loading, and in some cases, development schedule and associated risk.

WICOMO (Wang Institute Cost Model) developed at the wang institute, and **DECplan** developed by Digital Equipment Corporation are automated estimation tools that are based on COCOMO.

Each of the tools requires the user to provide preliminary 20 c estimates. These estimates are categorized by programming language and type (i.e., adapted code, reused code, new code). The user also specifies values for the cost driver attributes.

Each of the tools produces estimated elapsed project duration (in months), effort in staff-months, average staffing per month, average productivity in LOC/pm, and cost per month.

SLIM is an automated costing system based on the Rayleigh-Putnam model SLIM applies the Putnam software model, linear programming, statistical simulation, and program evaluation and review technique, or PERT techniques to derive software project estimates.

Once software size has been established, SLIM computes size deviation, a sensitivity profile that indicates potential deviation of cost and effort, and a consistency check with data collected for software systems of similar size.

The planner can invoke a linear programming analysis that considers development costraints on both cost and effort and provides a month-by-month distribution of effort and a consistency check with data collected for software systems of similar size.

Q.46. Describe COCOMO model. ***(R.G.P.V., June 2006, Dec. 2014)***

Or

How COCOMO model work for the cost estimation ?

(R.G.P.V., Dec. 2015)

Or

Explain cost estimation COCOMO model with exmple.

(R.G.P.V., June 2016)

Ans. Instead of being a function of one variable, resource estimates can depend on many different factors. Thus, giving rise to multivariable models. One way of building multivariable models is to begin with an starting estimate determined by using the static single-variable model equations, that depend on size, and then adjusting the estimates based on other variables. This approach indicates that size is the primary factor for cost, while other factors have a lesser effect. One such model called the COnstructive COst MOdel (COCOMO), was developed by Boehm. This model also estimates the total effort (person-months) of the technical project staff, which includes development, management, and support tasks, but does not include the cost of secretarial and other staff, that might be needed in the firm.

The basic steps of COCOMO model are as follows –

(i) Obtain a starting estimate of the development effort from the estimate of KLOC (1000, LOC).

(ii) Determine a set of 15 multiplying factors from different project attributes.

(iii) Adjust the effort estimate by multiplying the starting estimate with all the multiplying factors.

The starting estimate, also called nominal estimate, is determined by an equation of the form of static single-variable models by using KLOC as the size measure. Now, to obtain initial effort E_i (person-months), the equation used is of the following type –

$$E_i = A \times (KLOC)^C$$

The value of the constants A and C depends on the project type. In COCOMO, the projects are divided into three categories, i.e., organic, semidetached, and embedded. ***Organic projects*** are in the area where the organization has considerable experience and requirements are less stringent. Such systems are developed by small teams. Examples are – simple business systems, simple inventory management systems, and data processing systems. ***Embedded projects*** are ambitious and novel. Here, the organization has little experience and the requirements are stringent. These systems have tight constraints from the environment (i.e., software, hardware, and people). Examples are – embedded avionics systems and real-time command systems. The ***semidetached systems*** come in between these two types. Examples are - developing a new operating system, a database management system, and a complex inventory management system.

As we have noticed previously, the values of the constants for a cost model depend on the process and thus have to be determined from the historical data about the process usage. COCOMO has instead provided "global" constant values. This might be successful because it requires some characteristics about the process, and it provides a way to fine-tune the estimate for a particular process.

Still, there is little data to support that these constants are globally applicable, and to apply COCOMO in an environment, these may need to be tailored for the specific environment. One approach is to begin with the COCOMO-supplied constants, until data for some completed projects are available.

By this method, the total cost of the project can then be easily estimated. For the purposes of planning and monitoring, effort estimates required for different phases are also needed. In COCOMO, effort for phase is considered a defined percentage of the total effort, which varies with the project type and size. Besides phase-wise estimates, the model can be used to estimate the cost of different components or subsystems of the system.

COCOMO provides three levels of models of increasing complexity i.e., basic, intermediate, and detailed. Among them, the detailed model is the most complex one. It has different multiplying factors for the different phases for a given cost driver. The system or module applicable cost drivers' set is also not the same as the drivers for the system level. However, it might be too lengthy for several applications. COCOMO also provides support in finding the rating of different attributes and performing sensitivity and trade-off analysis.

Example – Suppose a system is designed for office automation, and it is clear that there will be four modules in the system i.e., data entry, data update, query and report generator. These are the requirements.

The sizes for the modules and complete system is estimated as –

Data entry	0.6 KDLOC
Data update	0.6 KDLOC
Query	0.8 KDLOC
Reports	1.0 KDLOC
Total	3.0 KDLOC

By these requirements, ratings of different cost drivers attributes are assessed. Ratings are (with factors) as –

Complexity	High	1.15
Storage	High	1.06
Experience	Low	1.13
Programmer capability	Low	1.17

All other factors had a nominal rating. From these, the effort adjustment factor is –

$$EAF = 1.15 * 1.06 * 1.13 * 1.17 = 1.61$$

The initial effort of planning can be obtained from equations, then we have

$$E_i = 3.2 * 3^{1.05} = 10.14 \text{ PM}$$

By using EAF, the effort estimate is –

$$E = 1.61 * 10.14 = 16.3 \text{ PM}$$

Table 5.4 Phase-wise Distribution of Effort

Phase	*Size*			
	Small 2 KDLOC	*Intermediate 8 KDLOC*	*Medium 32 KDLOC*	*Large 128 KDLOC*
Product design	16	16	16	16
Detailed design	26	25	24	23
Code and unit test	42	40	38	36
Integration and test	16	19	22	25

By using table 5.4 of phase-wise distribution, (which is given above) we can get percentage of total effort that is consumed in different phases. The office automation system's size is 3 KDLOC, so by using interpolation we can get appropriate percentage. Consider the percentage of different phases are –

Design (16%), detailed design (25.83%), code and unit test (41.66%) and integration and testing (16.5%). Then effort estimates are –

System Design	0.16 * 16.3 = 2.6 PM
Detailed Design	0.258 * 16.3 = 4.2 PM
Code and Unit Test	0.4166 * 16.3 = 6.8 PM
Integration	0.165 * 16.3 = 2.7 PM.

Q.47. Briefly describe various types of COCOMO models.

Or

Discuss cost estimation models. ***(R.G.P.V., Dec. 2003)***

Or

Discuss various cost estimation models and also compare them.
(R.G.P.V., June 2010, 2011, 2015)

Ans. Barry Boehm introduces a hierarchy of software estimation models bearing the name COCOMO, for COnstructive COst MOdel. Boehm's hierarchy of models takes the following form –

Model 1 The Basic COCOMO model computes software development effort (and cost) as a function of program size expressed in estimated lines of code.

Model 2 – The intermediate COCOMO model computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.

Model 3 – The Advanced COCOMO model incorporates all characteristics of the intermediate version with as assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

The basic COCOMO equations take the form –

$$E = a_b KLOC^{b_b}$$

$$D = c_b E^{d_b}$$

where E is the effort applied in person-months, D is the development time in chronological months, and KLOC is the estimated number of delivered lines of code for the project (express in thousands). The coefficient a_a and c_b and the exponents b_b and d_b are shown in table 5.5.

Table 5.5 Basic COCOMO Model

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

The basic model is extended to consider a set of "cost driver attributes" that can be grouped into four major categories – product attributes, hardware attributes, personnel attributes, and project attributes. Each of the 15 attributes in these categories is rated on a six-point scale that ranges from "very low" to "extra high" (in importance or value). Based on the rating, an effort multiplier is determined from tables published by Boehm, and the product of all effort multipliers results in an effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4.

The intermediate COCOMO model takes the form –

$$E = a_i \; KLOC^{b_i} \times EAF$$

where E is the effort applied in person-months and KLOC is the estimated number of delivered lines of code for the project. The coefficient a_i and the exponent b_i are given in table 5.6.

Table 5.6 Intermediate COCOMO Model

Software Project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

The value of project duration enables the planner to determine a recommended number of people, N, for the project –

$$N = E/D$$

Q.48. Write short note on project scheduling.
(R.G.P.V., Dec. 2006, June 2012, 2013, Dec. 2014)

Ans. After the cost estimation, the schedule estimation and staff requirement estimation may be the most important activities. If phase-wise cost is available then both activities are related to each other.

The main aim of the schedule estimation is to determine the total duration of the project and the duration of the different phases. A schedule cannot be simply obtained from the total effort estimate which deciding by on average size and then finding the total time requirement which dividing by the overall effort by the average staff size.

So, for completing the project, there is some relationship required between the project duration and the total effort in person-months. But this relationship is not linear i.e., the project duration to halve, doubling the staff-months will not work. Behind this there is a basic reason which is that if the staff needs to communicate for the completion of the task, then the communication time should be accounted for. With the square of the number of staff increases the communication time.

Now, it is clear that we cannot treat the schedule, as a variable totally in control of management. Some time will be required by every project to finish and that time cannot be reduced putting some more people on the project. So, it is clear that the project schedule must be assessed for planning and it is an independent variable. There are following models which are mostly used to assess the project duration.

(i) Average Duration Estimation – To determine the total duration of the project then we can use single variable models. Normally, schedule is modeled as depending on the total effort. Again, the constants for model are determined from the old data. The total duration found by the IBM Federal System Division, M, in calendar months can be estimated by –

$$M = 4.1 \; E^{0.36}$$

The schedule is find out in a same was in COCOMO model. The equation for an organic type of software is –

$$M = 2.5 \; E^{0.38}$$

The constants vary slightly for the other projects types. Obtaining the duration or schedule of the different phase is in the same way as in the effort distribution. The different phases percentages are shown in table 5.7 as follows –

Table 5.7 Phase-wise Distribution of Schedule

Phase	Size			
	Small 2 KDLOC	*Intermediate 8 KDLOC*	*Medium 32 KDLOC*	*Large 128 KDLOC*
Product Design	19	19	19	19
Programming	63	59	55	51
Integration	18	22	26	30

In the above COCOMO table, the detailed design, coding and the unit testing phase are included into one programming phase, because all these activities are generally done by the programmers. The system design and integration are often done by different people. People may or may not be involved in programming activities of the project. In detailed design, coding and unit testing, the percentage of total project duration spent, which somewhat higher than the percentage of total effort spent in these activities. The percentages are slightly different, which means that the average staff sizes which computed for the different phases will not be much different from the total average staff size for the project.

(ii) Project Scheduling and Milestones – After the estimates of the effort and time requirement for the different phases, then we can be prepared the schedule for the project. Such scheduling will be used to monitor the progress of the project later.

The Gantt chart is a conceptually simple and effective scheduling technique which uses a calendar-oriented chart for representing the project schedule. As a bar each activity is represented in the calendar, which start from the date of starting of the activity and ending with the ending date of that activity. The start end of each activity in the chart becomes milestone for the project.

In a Gantt chart, we can easily represents the progress, by coloring or ticking off each milestone when it is completed. On the other hand, another bar can be drawn for each activity which specifying when the activity actually started and ended, that means when these two milestones were actually achieved.

There is a main drawback with the Gantt chart is that it does not depict the dependency relationships among the different activities. Hence, the effect of slippage in one activity on the other activities or on the total project schedule cannot be determined. So, it is heavily used and it is conceptually simple and easy to understand. So, for small and medium-sized projects it is sufficient.

For large project PERT charts are often used. A PERT chart is a graph-based chart. It is used to represent the dependencies among different activities. The activities form the "critical path," which if delayed then overall project will delay. The use of this chart to justify in large project. The PERT chart is not conceptually as simple as Gantt chart and also the representation is graphically not as clear as Gantt charts.

Q.49. Discuss the principles of project scheduling.

Ans. There are following principles of project scheduling –

(i) Compartmentalization – It divides the project into several tasks. The purpose of compartmentalization is to make the project manageable. Thus, it becomes easier to prepare the project schedule according to these tasks.

(ii) Interdependency – Determines the interdependency of one or more activities or tasks on each other. All the activities of the project are not independent. There are various activities that are performed sequentially, whereas some of the activities are executed together with other activities on the other hand, some activities cannot begin until the activity on which they are dependent is complete.

(iii) Time Allocation – Determines the time to be allocated to each project management team member for performing specified activities. However, before allocating time, it is important to estimate the effort required by them to complete the assigned task. In addition, the project management team members should be assigned a start date and an end date according to the work to be conducted on full-time or part-time basis.

(iv) Effort Validation – Ensure that the efforts required to perform the assigned task are valid. In other words, it should verify that the task allocated to one or more project management team members is according to the effort required for each task. This is because every project management team has a defined number of team members.

(v) Defined Responsibilities – Specify the roles and responsibilities of every project management team member. Hence, the task should be allocated according to the skills and abilities of team members to perform the assigned task.

(vi) Defined Outcomes – Specify the outcomes of every task performed by the project management team members. The outcome is achieved after completion of a task. Generally, the outcome of a task is in the form of a product and these products are combined in deliverables.

(vii) Defined Milestones – Specify the milestones when work products are complete and reviewed for quality.

Q.50. Discuss the techniques of project scheduling in details.

Ans. The following are important techniques of project scheduling –

(i) Work Breakdown Structure (WBS) – A work breakdown structure is a hierarchic decomposition or breakdown of a project or major activity into successive levels, in which each level is a finer breakdown of the

preceding one. In final form, a WBS is very similar in structure and layout to a document outline. Each item at a specific level of a WBS is numbered consecutively (i.e., 10, 20, 30, 40, 50). Each item at the next level is numbered within the number of its parent item (i.e., 10.1, 10.2, 10.3, 10.4).

The process can be repeated until each goal is small enough to be well understood. Then each goal can be individually planned for its. Resource requirements, Assignment of responsibility, and scheduling etc. A four level WBS diagram based on project phases is shown in fig. 5.15.

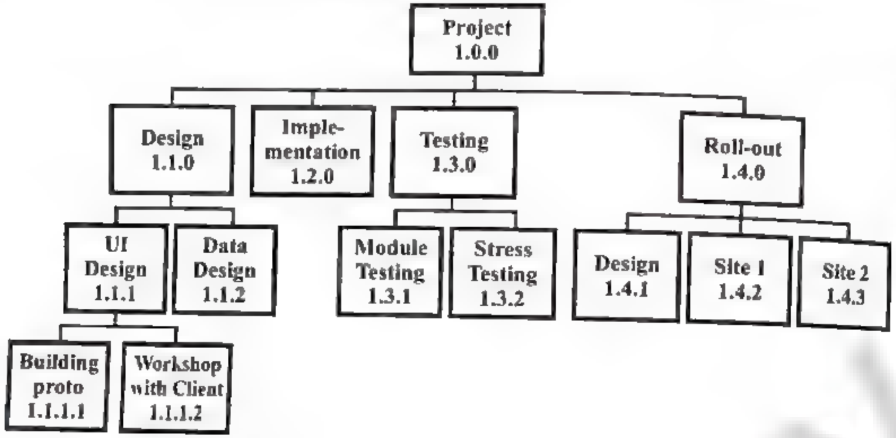


Fig. 5.15 Four Level WBS Diagram Based on Project Phases

(ii) Activity Charts or Activity Networks – The foundation of this technique originated from the special projects office of the U.S Navy in 1958. They developed this technique for evaluating the performance of large development projects. This tech- nique can be broken down into three stages –

(a) Planning – It identifies tasks and estimates duration of time. Also, it arranges in feasible sequence, and draw diagram.

(b) Scheduling – It establishes time-table of start and finish times.

(c) Analysis – It establishes float, evaluates and revises as necessary.

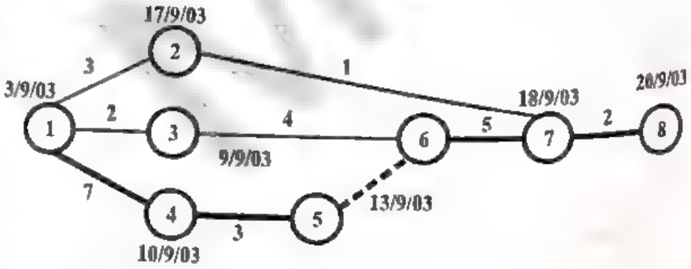


Fig. 5.16 Example of an Activity Network

The activity network of tasks needed to complete a project, showing the order in which the tasks need to be completed and the dependencies between them. This is represented graphically, which is shown in fig. 5.16.

The diagram consists of a number of circles, representing events within the development lifecycle, such as the start or completion of a task, and lines, which represent the tasks themselves. Each task is additionally labelled by its time duration. Thus the task between event 4 and 5 is planned to take 3 time units. The primary benefit is the identification of the critical path.

The critical path = Total time for activities on this path is greater than any other path through the network (delay in any task on the critical path leads to a delay in the project).

(iii) Project Evaluation Review Technique (PERT) – It is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT is a methodology developed by the U.S. Navy in the 1950s to manage the Polaris submarine missile program.

A similar methodology, the critical path method (CPM), which was developed, for project management in the private sector at about the same time, has become synonymous with PERT, so that the technique is known by any variation on the names, PERT, CPM or PERT/CPM. Unlike Gantt charts, PERT can be both a cost and a time management systems. PERT is organized by events and activities or tasks.

PERT is designed for research and development type projects when activity completion times are uncertain. The heart of any PERT chart is a network of tasks needed to complete a project, showing the order in which the tasks need to be completed and the dependencies between them.

PERT charts depict task, duration and dependency information. Each chart starts with an initiation node from which the first task, or tasks, originates. If multiple tasks begin at the same time, they are all started from the node or branch, or fork out from the starting point. Each task is represented by a line, which states its name or other identifier, its duration, the number of people assigned to it, and in some cases the initials of the personnel assigned. The other end of the task line is terminated by another node, which identifies the start of another task, or the beginning of any slack time, that is, waiting time between tasks.

(iv) Gantt Charts – A Gantt chart is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist. It is frequently used in project management.

A Gantt chart is useful for tracking and reporting progress, as well as for graphically displaying a schedule. Gantt charts are often used to report progress because they represent an easily understood picture of project status. However, Gantt charts are not an ideal tool for project control.

Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project. Project management tools incorporating Gantt charts include PRINCE, MacProject, Microsoft Project and Microsoft Excel.

The purpose of a Gantt chart is to present a project schedule that shows the relationship of activities over time. Gantt charts are a project planning tool that can be used to represent the timing of tasks required to complete a project. A simple Gantt chart is shown in fig. 5.17.

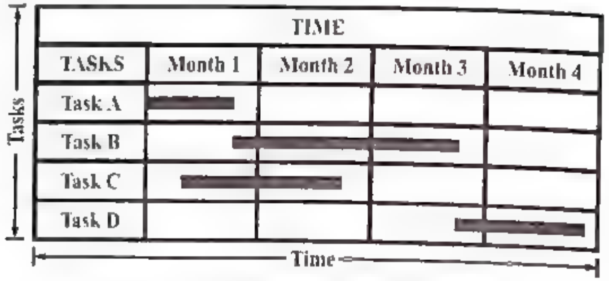


Fig. 5.17 Simple Gantt Chart

(v) Critical Path Method (CPM) – CPM charts are similar to PERT charts and are sometimes known as PERT/CPM. CPM acts as the basic both for preparation of a schedule and of resource planning. They were developed in the 1950s to control large defence projects and have been used routinely since then. During management of a project, it allows to monitor the achievements of project goals. It also helps to see where remedial action needs to be taken to get a project back on course.

In a CPM chart, the critical path is indicated. Critical path is the path of longest duration as determined on a project network diagram. The critical path determines the total duration of the project. If a task on the critical path is delayed, the final completion of the project will likely to delayed. The critical path is critical because tasks that follow a critical task cannot be started until all of the previous tasks on the critical path are completed. Thus, if a task on the critical path is delayed, all tasks following the delayed critical task will be pushed out in time. The critical tasks will have starting and finishing times that are fixed relative to the start of the project.

Q.51. Software project scheduling does not differ from scheduling of any other multitask engineering project. Discuss. (R.G.P.V., June 2017)

Ans. Software project scheduling does not differ largely from scheduling of any other multitask engineering project. As a results, generalized tools of project scheduling and methods are applied with some modification in software projects.

There are two project scheduling techniques, which can be applied to software development –

(i) Program evaluation and review technique (PERT).

(ii) Critical path method (CPM).

Both methods are operated by information obtained in project planning activities, which are given below –

(i) Decomposition of operation

(ii) Estimates of effort

(iii) The suitable process model and operation set are selected.

(iv) A decomposition of the product function.

A operation network defines the interdependencies among operations. In many cases, tasks (operations) known as the project work breakdown structure (WBS), can be expressed for the product as a complete or for individual functions.

Both PERT and CPM are implemented in a wide variety of automated tools, which are available for pc. It is easy to use. Boundary time calculations are very useful in software project scheduling

Refer to Q.48 and Q.50 (iii), (v).

Q.52. What are the relative advantages of using PERT charts and Gantt charts in scheduling and monitoring software project ? (R.G.P.V., June 2005)

***Ans.* Advantages of PERT** – There are following benefits of PERT –

(i) The PERT network is continuously useful to project managers prior to and during a project.

(ii) The PERT network is straightforward in its concept and is supported by software.

(iii) The PERT network's graphical representation of the project's tasks help to show the task interrelationships.

(iv) The PERT network's ability to highlight the project's critical path and task slack time allows the project manager to focus more attention on the critical aspects of the project-time, costs and people.

(v) The use of the PERT network is applicable in a wide variety of projects.

(vi) PERT is a scheduling tools that also shows graphically which tasks must be completed before other are begun.

(vii) By displaying the various task paths, PERT enables the calculation of a critical path.

(viii) PERT controls time and costs during the project and also facilitates finding the right balance between completing a project on time and completing it within the budget.

(ix) PERT forces the manager to plan.

(x) It exposes all possible parallelism in the activities and thus helps in allocating resources. Also it allows scheduling and simulation of alternative schedules.

Advantages of Gantt Charts –

(i) Provide an easily understandable overview of a project for those without any technical background.

(ii) Gantt charts are relatively easy to create and maintain.

(iii) Able to reflect the status of each project task at any point in time.

(iv) Able to represent overlapping or parallel task.

Q.53. What is Gantt chart ? Give the advantages of Gantt charts in monitoring software project. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.50 (iv) and Q.52.

Q.54. What is software project tracking ?

Ans. The purpose of software project tracking is to provide adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.

Software project tracking involves tracking and reviewing the software accomplishments and results against documented estimates commitments, and plans, and adjusting plans based on the actual accomplishments and results.

Q.55. How can software project tracking be accomplished ?

Ans. As the project progresses, the project manager understands the activities to be completed and milestones to be tracked and controlled with the help of project schedule. Tracking of project schedule is done in several ways –

(i) Conducting Periodic Meetings with Team Members – By conducting periodic meetings, the project manager is able to distinguish between completed and uncompleted activities or those that are yet to start. In addition, the project manager, considers the problems in the project as reported by the team members.

(ii) Assessing the Results of Reviews – Software reviews are conducted when one or more activities of the project are complete or when a particular development phase is complete. The purpose of conducting reviews is to check whether the software is developed according to user requirements or not.

(iii) Determining the Milestones – Milestones indicating the expected output are described. These milestones check the status of a project by comparing progress of activities with the estimated end data of the project.

(iv) Using Earned Value Analysis to Determine the Progress of the Project – The progress of the project is determined quantitatively by earned value analysis technique. This technique provides an estimate for every task without considering its type and the total hours required to accomplish the project. Based on this estimation, each activity is given an earned value, which is a measure of progress and describes the percentage of the activities that have been completed.

Q.56. What is risk management ?

Ans. Risk is defined as an exposure to the chance of injury or loss. That is, risk implies that there is a possibility that something negative may happen. In the context of software projects, negative implies that there is an adverse effect on cost, quality, or schedule. Risk always involves two characteristics –

(i) Uncertainty – The risk may or may not happen, that is, there are no 100% probable risks.

(ii) Loss – If the risk becomes a reality, unwanted consequences or losses will occur.

When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk.

Q.57. Write a brief note on risk assessment. ***(R.G.P.V., June 2014)***

Ans. Risk assessment concentrates on determination of risks. The occurrence of risks depends on their nature, scope, and timing. Nature of risks specifies the problems that arise when risk occurs. Scope of risks specifies the capability of risks to impact the software. Timing of risks considers when and for how long the affect of risk is observed. With the help of proper risk assessment, risks can be prevented. With complete information of risk assessment, the probability of occurrence of risks and its severity is determined. Generally, risk assessment can be carried out at any time during the project. To effectively perform risk assessment, the management should consider the occurrence of risks instead of assuming that no risks will occur during software development. In addition, a record of risk assessment should be maintained.

Q.58. What do you mean by risk mitigation ? ***(R.G.P.V., Dec. 2014)***

Ans. Risk mitigation or reducing the impact of risk is done through an RMMM plan. An RMMM plan deals with mitigation, monitoring and management of risks in a systematic manner. The steps in an RMMM plan are as under. Table 5.8 shows a sample case of risk and its mitigation plan.

Table 5.8 Typical Table Showing Considered Risk and RMMM Plan

Risk Particulars			***RMMM***		
Provisional Risk Name	***Risk ID***	***Severity Impact Index***	***Strategy***	***Plan Details***	***Review***
Volatility of Requirement specifications due to end users	6	6	Prevention	(a) Prototype (b) Proof of concept experimentation	Prototype implantation Proof of concept confirmation
Inexperience and lack of skills	22	4	Prevention	(a) Seminar on domain knowledge (b) Training on skills	End of seminar and training courses and revisiting RMP
Staff turnover	9	9	Acceptance	No plan of action	—
Application knowledge absent	3	6	Transfer	Outsource the development	Periodical

(i) Prepare a prioritised risk list based on risk exposure and its severity index.

(ii) Determine risk resolution strategy for each.

(iii) Design an action plan based on resolution strategy to deal with risk.

(iv) Institute a monitoring plan through systematic review, and MIS reports on risks integrated into software development MIS reports.

(v) Take corrective measures to control the impact.

Q.59. Explain risk assessment and mitigation. ***(R.G.P.V., June 2016)***

Ans. Refer to Q.57 and Q.58.

Q.60. What are risk management activities ? ***(R.G.P.V., June 2011)***

Ans. Risk management process has several activities –

(i) Risk Assessment – Risk assessment include the following –

(a) Risk Identification – Risk identification is a systematic attempt to specify threats to the project plan. The purpose of risk identification is to develop a list of risk items called risk statement. Risk identification can be facilitated with the help of a checklist of common risk areas for software projects, or by examining the contents of an organizational database of previously identified risks and mitigation strategies.

(b) Risk Analysis – When the risk have been identified, all items are analyzed using different criteria. The purpose of the risk analysis is to assess the loss probability and magnitude of each risk item. The input is the risk statement and context developed in the identification phase. The output of this phase is a risk list containing relative ranking of the risks and a further analysis of the description, probability, consequence, and context.

(c) Risk Prioritization – It helps the project focus on its most severe risks by assessing the risk exposure. Exposure is the product of the probability of incurring a loss due to the risk and the potential magnitude of that loss.

(ii) Risk Control – Risk control is the process of managing risks to achieve the desired outcomes. Risk control process involves the following –

(a) Risk Planning – Risk management planning produces a plan for dealing with each significant risk, include mitigation approaches, owner and timelines. Risk planning is to identify strategies to deal with risk. These strategies are – risk avoidance, risk minimization, and risk contingency plans.

(b) Risk Mitigation – The risk mitigation is a plan that would reduce or eliminate the highest risks. The mitigation plan includes a description of the actions that can be taken to mitigate the red rated risks and assigns a primary handler for the action.

(c) Risk Resolution – Risk resolution is the execution of the plans for dealing with each risk. If the risk is at the watch list, a plan of how to resolve the risk already has taken place. The project manager has to respond to the already chalked out plan of how to resolve the risk.

(d) Risk Monitoring – Risk monitoring is the continually reassessing of risks as the project proceeds and conditions change.

(iii) Risk Reporting – Risk reporting is reporting the status of the risks that were identified during risk identification and assessment stages. All types of risks along with their status are reported properly as part of risk reporting activity. The entire information about risks is documented together with the full history of risks such as the name of the risks, a risk statement, context, etc.

Q.61. What is risk management ? Explain the essential activities of risk management. ***(R.G.P.V., Dec. 2010)***

Ans. Risk management is the area that tries to ensure that the impact of risks on cost, quality and schedule is minimal.

The main aim of risk management is to address the problem of identifying and managing the risks associated with a software project. To anticipate risks which might affect the project schedule or the quality of the software being developed and to take action to avoid the risk, is an important task of a project manager. The risks identifying and drawing up plans to minimize their effect on the project is called risk management. To avoid disasters or heavy losses is the basic motivation of risk management.

The risk can be categorised as follows –

(i) Project Risks – These are the risks which affect the project schedule or resources.

(ii) Product Risks – These are the risks which affect the quality or performance of the software being developed.

(iii) Business Risks – These are the risks which affect the organization developing or procuring the software.

This classification is not an exclusive classification. If an experienced programmer leaves a project then it is project risk because the delivery of the system may be delayed, a product risk because a replacement may not be an experienced and so may make mistakes and a business risk because that experience is not available for bidding for future business.

For software projects the risk management is very important because of the inherent uncertainties, which most projects face. The process of risk management is shown in fig. 5.18.

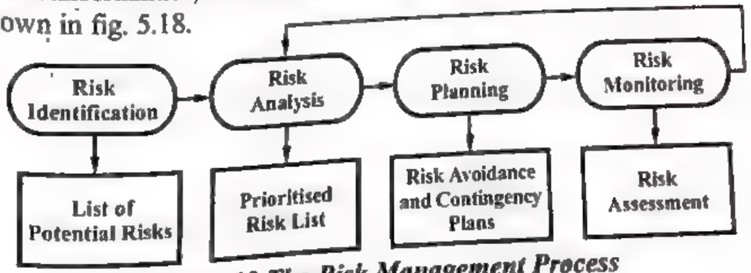


Fig. 5.18 The Risk Management Process

Table 5.9 Top-10 Risk Items and Techniques for Managing them

S.No.	Risk Item	Risk Management Techniques
(i)	Personnel shortfalls	Staffing with top talent; Job matching; Teambuilding; Key-personnel agreement; Training; Prescheduling key people
(ii)	Unrealistic schedules and budgets	Detailed multisource cost and schedule estimation; Design to cost; Incremental development; Software reuse; Requirements scrubbing
(iii)	Developing the wrong software functions	Organization analysis; Mission analysis; OPS-concept formulation; User surveys; Prototyping; Early users' manuals
(iv)	Developing the wrong user interface	Prototyping; Scenarios; Task analysis; User characterization (functionality, style, workload)
(v)	Gold plating	Requirements scrubbing; Prototyping; Cost-benefit analysis; Design to cost
(vi)	Continuing stream of requirements changes	High change threshold; Information hiding; Incremental development (defer changes to later increments)
(vii)	Shortfalls in externally furnished components	Benchmarking; Inspections; Reference checking; Compatibility analysis
(viii)	Shortfalls in externally performed tasks	Reference checking; Preaward audits; Award-fee contracts; Competitive design or prototyping; Teambuilding
(ix)	Real-time performance shortfalls	Simulation; Benchmarking; Modeling; Prototyping; Instrumentation; Tuning
(x)	Straining computer science capabilities	Technical analysis; Cost-benefit analysis; Prototyping; Reference checking

Risk Management Activities – Refer to Q.60.

Q.62. List and explain the steps in risk management process.
(R.G.P.V., May 2019)

Ans. Refer to Q.60 and Q.61.

Q.63. Suppose you are the project manager of a large software development project. List three common types of risks that your project might suffer. Point out the main steps that you would follow to effectively manage risks in your project. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.61.

Q.64. Explain the primary responsibility of software quality assurance group. ***(R.G.P.V., Dec. 2017)***

Ans. The primary responsibility of a software quality assurance group is to ensure that the required level of product quality is to be attained in an organization. The principle of quality management involves the procedures and standards which should be used during software development. All the cases of checking should be followed by the engineers. This shows the quality management of the system. The quality managers helps to develop the good quality product where everyone will be responsible for the development of the product and committed to attained a high level quality.

The quality assurance group encourage development team to take responsibility for the quality of their work and to develop the new approaches to improve the quality. The quality managers recognise the standards and procedures of quality management and the intangible aspects of software quality. Among the all team members, the interested candidates were supported in these intangible aspects of quality and encourage their professional behaviour.

There are three principle activities of software quality management –

(i) Quality Assurance – This principle establish a framework of organizational procedures and standards which lead to a high-quality software.

(ii) Quality Planning – This principle is used to select a suitable procedures and standards from this framework and to adapt the specific software project.

(iii) Quality Control – According to this principle the project quality procedures and standards processes are ensured by the software development team.

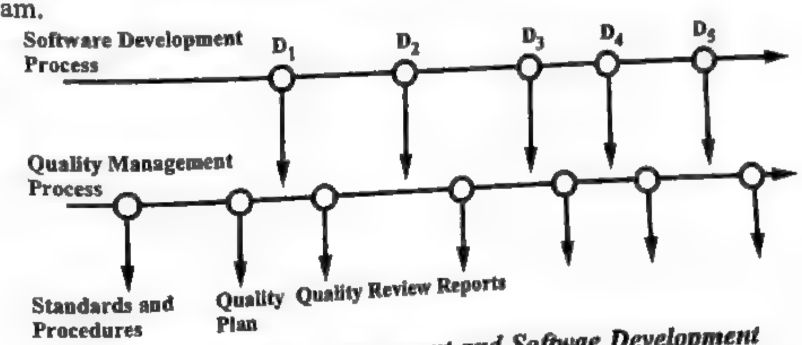


Fig. 5.19 Quality Management and Softwae Development

Q.65. What is software quality assurance ? Explain major SQA activities. ***(R.G.P.V., June 2010)***

Or

Write short note on software quality assurance (SQA). ***(R.G.P.V. Dec. 2009, June 2012)***

Or

Describe software quality assurance (SQA). ***(R.G.P.V., Dec. 2016)***

Or

What are the various quality concepts of SQA ? ***(R.G.P.V., May 2018)***

Ans. Software quality is defined as the quality that ensures customer satisfaction by offering all the customer deliverables on performance, standards, and ease of operations. Software quality assurance (SQA) is ensured through a quality management system (QMS). QMS is made of several components; it is a system integrated in the bigger system of software development, which comprises project, process, and product management systems. Software quality assurance is made of number of tasks associated with two different constituencies – the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting.

Software engineers address quality by applying solid technical methods and measures, conducting formal technical reviews, and performing well planned software testing.

The charter of the SQA group is to assist the software team in achieving a high quality end product. The software engineering institute recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting. These activities are done by an independent SQA group that –

(i) Prepares an SQA Plan for a Project – The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies.

(a) Evaluations to be performed.
(b) Standards that are applicable to the project.
(c) Audits and reviews to be performed.
(d) Procedures for error reporting and tracking.
(e) Documents to be produced by the SQA group.
(f) Amount of feedback provided to the software project team.

(ii) Participates in the Development of the Project's Software Process Description – The software team selects a process for the work to be done. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards, and other parts of the software project plan.

(iii) Reviews Software Engineering Activities to Verify Compliance with the Defined Software Process – The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

(iv) Audits Designated Software Work Products to Verify Compliance with those defined as Part of the Software Process – The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

(v) Ensures that Deviations in Software Work and Work Products are Documented and Handled according to a Documented Procedure – Deviations may be encountered in the project plan, process description, applicable standards, or technical work products.

(vi) Records any Noncompliance and Reports to Senior Management – Noncompliance items are tracked until they are resolved.

Moreover, SQA group also coordinates the control and management of change and helps to collect and analyze software metrics.

Q.66. What are the activities performed during SQA ? ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.65.

Q.67. What do you understand by software project management ? How quality of software can be assured ? ***(R.G.P.V., June 2014)***

Ans. Refer to Q.26 and Q.65.

Q.68. What is software quality assurance ? What are the measures of software quality ? ***(R.G.P.V., Dec. 2017)***

Ans. **SQA** – Refer to Q.65.

Software measurement is concerned by the numeric value which was derived for the attributes of the software product or the software process. On the comparisons of these values with each other, it is possible to draw the conclusions about the quality of software or the software process.

It is difficult to measure the software quality attributes directly. There are different factors which are used to measure or to predict the software product quality such as maintainability, complexity and understandability. Assume that some internal relationship exists between the attribute of the software, which we can measure and want to know. There should be a clear and validated relationship between the internal and the external software attributes.

Some of the factors which shows the measure of software quality are as follows –

(i) The internal attribute must be measured accurately.

(ii) A relationship must exist between measure and the externa behavioural attribute.

(iii) There should be understandable relationship, which can t validate and can be expressed in terms of a formula or model.

The fig. 5.20 shows the external quality attributes which might be related to the interest of internal attributes, and these attributes can be measured and relate to the external attribute.

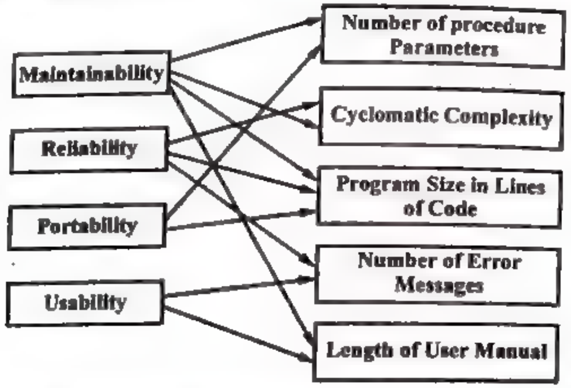


Fig. 5.20 Relationships between Internal and External Software Attributes

Q.69. What do you understand by the project plan ?

Ans. Each step in the software engineering process should produce a deliverable that can be reviewed and that can act as a foundation for the steps that follow. The software project plan is produced at the culmination of the planning tasks. It provides baseline cost and scheduling information that will be used throughout the software process.

The software project plan is a relatively brief document that is addressed to a diverse audience. It must (i) communicate scope and resources to software management, technical staff, and the customer; (ii) define risks and suggest risk aversion techniques; (iii) define cost and schedule for management review; (iv) provide an overall approach to software development for all people associated with the project; and (v) outline how quality will be ensured and change will be managed.

A presentation of cost and schedule will vary with the audience addressed. If the plan is used only as an internal document, the results of each estimation technique can be presented. When the plan is disseminated outside the organization, a reconciled cost breakdown (combining the results of all estimation techniques) is provided. Similarly, the degree of detail contained within the schedule section may vary with the audience and formality of the plan.

It is important to note that the software project plan is not a static document. That is, the project team revisits the plan repeatedly – updating risks, estimates, schedules and related information – as the project proceeds and more is learned.

Q.70. Explain project metrics. ***(R.G.P.V., Dec. 2005)***

Ans. Software project metrics are used for strategic purposes. Project metrics and indicators derived from software project measures are used by a project manager and its team to adapt project work flow and technical activities.

The first application of project metrics on most software projects occurs during estimation. Metrics collected from past projects are used as a basis from which effort and time duration estimates are made for current software work. As a project proceeds, measures of efforts and calendar time expanded are compared to original estimates. The project manager uses these data to monitor and control progress.

As technical work commences, other project metrics begin to have significance. Production rates represented in terms of pages of documentation, review hours, function points, and delivered source lines are measured. In addition, errors uncovered during each software engineering tasks are tracked. As the software evolves from specification into design, technical metrics are collected to access design quality and to provide indicators that will influence the approach taken to code generation and module and integration testing.

The intent of project metrics is two fold. First, these metrics are used to minimize the development schedule by guiding the adjustments necessary to avoid delays and mitigate potential problems and risks. Second, project metrics are used to access product quality on an ongoing basis and when necessary, modify the technical approach to improve quality.

As quality improves, errors are minimized, and as the errors count goes down, the amount of rework required during the project is also reduced. This leads to a reduction in overall project cost.

Another model of software project metrics suggests that every project should measure –

(i) ***Inputs*** – Measures of the resources (e.g., people, environment) required to do the work.

(ii) ***Outputs*** – Measures of the deliverables or work products created during the software engineering process.

(iii) ***Results*** – Measures that indicate the effectiveness of the deliverables.

In actuality, this model can be applied to both process and project. In the project context, the model can be applied recursively as each framework activity occurs. Therefore, the outputs from one activity become inputs to the next. Results metrics can be used to provide an indication of the usefulness of work products as they flow from one framework activity to the next.

Q.71. Write short note on project plan and metrics. ***(R.G.P.V., June 2012)***

Ans. Refer to Q.69 and Q.70.

NUMERICAL PROBLEMS

Prob.1. What are the different categories of software according to COCOMO estimation model ? Suppose that, you are developing a software product in the organic mode. You have estimated size of product to be about 1,00,000 LOC. Compute nominal effort and development time.

(R.G.P.V., June 2003, Dec. 2006)

Sol. **Categories of Software According to COCOMO Estimation Model** - Refer to Q.46.

As per problem, Size = 1,00,000 LOC = 100 KDLOC

For organic mode, a = 3.2 and b = 1.05

$\therefore$ Nominal effort, $E_i = a \times (\text{Size})^b$

$= 3.2 \times (100)^{1.05}$

$= 3.2 \times 125.8925412$

$= 402.8561318$ PM

Hence, nominal effort = **402.9 PM** **Ans.**

Minimum development time is given by

$M = 2.5E^{0.38}$

$\therefore$ $M = 2.5 \times (402.9)^{0.38}$ months

$= 2.5 \times 9.77$

$= 24.43$

$\therefore$ Development time = **24.43 months** **Ans.**

Prob.2. A project size of 2000 KLOC is to be developed. Software development team has average experience on similar types of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project. (R.G.P.V., Dec. 2015)

Sol. The project size is 2000 KLOC.

For semi-detached mode, a = 3.0, b = 1.12, c = 2.5 and d = 0.35

Effort (E) $= a \times (\text{size})^b$

$= 3.0 \times (2000)^{1.12}$

= **14937.39 PM** **Ans.**

Development time (D) $= c \times (E)^d$

$= 2.5 \times (14937.39)^{0.35}$

= **72.26 M** **Ans.**

Average staff size (SS) = E/D

= 14937.39/72.26

= 206.71 P **Ans.**

Productivity (P) = **KLOC/E**

= 2000/14937.39

= **0.1338** **Ans.**

Prob.3. Consider a project to develop a full screen editor. The major components identified are –

(i) Screen edit
(ii) Command language interpreter
(iii) File input and output
(iv) Cursor movement
(v) Screen movement

The sizes estimasted are 4 K, 2 K, 1 K, 2 K and 3 K delivered source lines of code. Use COCOMO model to determine –

Overall cost and schedule estimates, assume effort adjustment factor being 1.21.

μp (effort) *τ_p (time)*	*System Design*	*Detail Design*	*Module Code and Test*	*Integration and Test*
	0.16 *0.19*	*0.26* *0.24*	*0.42* *0.39*	*0.16* *0.18*

(R.G.P.V., Nov./Dec. 2007)

Sol. ***(i) Overall Cost and Schedule Estimates*** – Screen editor may be classified into organic type as it is a data processing system, which requires considerable experience, less stringent requirements and a small team.

The sizes for the different modules are estimated to be –

(i) Screen edit	4 KDLO
(ii) Command language interpreter	2 KDLO
(iii) File input and output	1 KDLO
(iv) Cursor movement	2 KDLO
(v) Screen movement	3 KDLO
Total	12 KDL

It has been assumed that effort adjustment factor being 1.21. That EAF = 1.21.

The initial effort estimate for the project is obtained from the relev equations. We have,

$E_i = a^*(\text{KDLOC})^b$

$= 3.2 * (12)^{1.05}$

$= 43.48$ PM

Using the EAF, the adjusted effort estimate is

$E = \text{EAF}^*E_i$

$= 1.21 * 43.48$

= **52.61 PM**

The overall size estimate for the project is 12 K and the final effort estimate obtained is 52.61 PM. As this is an organic system, the overall duration or schedule will be determined by the equation $2.5*E^{0.38}$. Using this, we get the project duration D as

$$D = 2.5*(52.61)^{0.38}$$

$$= \mathbf{11.27\ months}$$ **Ans.**

(ii) Cost and Schedule Estimate for Different Phases – The percentages for the total effort consumed in different phases are – system design – 16%, detailed design – 24.83%, code and test – 39.67%, and integration and testing – 19.5%. With these, the effort estimates for the different phases are –

System design	0.16 * 52.61 = 8.42 PM
Detailed design	0.24 * 52.61 = 12.63 PM
Code and test	0.39 * 52.61 = 20.52 PM
Integration and test	0.19 * 52.61 = 9.99 PM

Using the table, which gives the duration of the different phases as a percentage of the total project duration, the duration of the different phases is –

System design	0.19 * 11.27 = 2.14 M
Programming	0.58 * 11.27 = 6.54 M
Integration	0.22 * 11.27 = 2.48 M

●●●

RGPV

B.E. (Sixth Semester) EXAMINATION, June 2012
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERING AND PROJECT MANAGEMENT
(CS-603)

Note : Attempt any *five* questions. All questions carry equal marks.

1. (a) Distinguish between generic and customized software products. Which one would generate more revenue for a company ? Give the reasons behind your answer. **(See Unit-I, Page 3, Q.2)** 10

(b) Explain prototype model. What are the advantages of developing a prototype of a system ? **(See Unit-I, Page 22, Q.27)** 10

Or

2. (a) Differentiate between a software product and a software process. **(See Unit-I, Page 11, Q.11)** 10

(b) Explain how a software development effort is initiated and finally terminated in the spiral model. **(See Unit-I, Page 26, Q.34)** 10

3. (a) What are the different types of requirements gathering activities that the analysts use to gather requirements from a customer ? 10 **(See Unit-II, Page 60, Q.9)**

(b) Briefly outline the important steps involved in developing a software system using a popular object oriented design methodology. 10 **(See Unit-IV, Page 201, Q.61)**

Or

4. (a) What do you understand by traceability in the context of software requirements specification ? How is traceability achieved ? 10 **(See Unit-II, Page 97, Q.50)**

(b) What are the important issues which an SRS document must address ? **(See Unit-II, Page 92, Q.45** 10

5. (a) Compare relative advantages of the object oriented and function oriented approaches to software design. **(See Unit-IV, Page 200, Q.64)** 10

(b) Discuss the different classification of architectural styles with respect to software and discuss each style in detail. 10 **(See Unit-III, Page 115, Q.26)**

Or

6. (a) Explain how the overall cohesion and coupling of a design would be impacted if all modules of the design are merged into a single module. 10

(b) What are the main shortcomings of Data Flow Diagram (DFD) as a tool for performing structured analysis ? (See Unit-II, Page 66, Q.16) 10

7. (a) Distinguish between an error and a failure in the context of program testing. Justify your answer. (See Unit-IV, Page 158, Q.10) 10

(b) What do you understand by system testing ? What are the different kinds of system testing that are usually performed on large software products ? (See Unit-IV, Page 186, Q.45) 10

Or

8. (a) What is test plan ? Write down the components of test plan and their purpose. (See Unit-IV, Page 190, Q.51) 10

(b) Explain various approaches of testing and also explain their testing methods. (See Unit-IV, Page 178, Q.36) 10

9. (a) What do you mean by the term software re-engineering ? Why is it required ? (See Unit-V, Page 214, Q.14) 10

(b) What is meant by Software Configuration Management ? (See Unit-V, Page 207, Q.5)

Why is software configuration management crucial to the success of large software product development projects ? 10

Or

10. Write short notes on the following : 5 each

(i) Software Quality Assurance (SQA) (See Unit-V, Page 257, Q.65)

(ii) Project Plan and Metrics (See Unit-V, Page 261, Q.71)

(iii) Reverse Engineering (See Unit-V, Page 221, Q.20)

(iv) Project Scheduling. (See Unit-V, Page 245, Q.48)

RGPV

B.E. (Sixth Semester) EXAMINATION, June 2013
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERING AND PROJECT MANAGEMENT
(CS-603)

Note : Attempt all questions. All questions carry equal marks.

1. (a) Explain how do the use of software engineering principles help to develop software products cost effectively and timely. Elaborate your answer by using suitable example. (See Unit-I, Page 6, Q.5) 10

(b) What do you mean by a software process ? What is the difference between a methodology and a process ? Explain using suitable examples. (See Unit-I, Page 7, Q.7) 10

Or

2. Consider the assertion : The classical waterfall model is an idealistic model.

(i) Justify why the above assertion is true.

(ii) Even if the classical waterfall model is an idealistic model, is there any practical use of this model at all ? Explain your answer. 20 (See Unit-I, Page 17, Q.19)

3. (a) What do you mean by system specification, requirement specification, software specification ? Explain in brief. 10 (See Unit-II, Page 87, Q.37)

(b) Differentiate between the following : 10

(i) Structural analysis and object oriented analysis. (See Unit-IV, Page 201, Q.63)

(ii) Data flow model and control flow model. (See Unit-III, Page 118, Q.27)

Or

4. (a) List five desirable characteristics of a good software requirement specification (SRS) document. (See Unit-II, Page 87, Q.36) 10

(b) Explain analysis modelling. What are its elements ? Also explain functional modelling. (See Unit-II, Page 68, Q.20) 10

5. (a) Differentiate between structured analysis and structured design in the context of function oriented design. (See Unit-III, Page 137, Q.59) 10

(b) Enumerate the different types of coupling that might exist between the two modules. Give examples of each. 10 (See Unit-III, Page 131, Q.48)

Or

6. (a) What do you understand by the term top down decomposition in the context of function oriented design ? Explain your answer with suitable example. (See Unit-III, Page 126, Q.40) 10

(b) What do you understand by design review ? What kinds of mistakes are normally pointed out by the reviewers ? 10 (See Unit-III, Page 110, Q.16)

7. (a) How can you compute the cyclomatic complexity of a program ? How is it useful in program testing ? (See Unit-IV, Page 188, Q.49) 10

(b) What is meant by code walkthrough ? What are some of the important types of errors checked during code walkthrough ? 10 (See Unit-IV, Page 155, Q.6)

Or

8. (a) Differentiate between blackbox testing and white box testing. 10 (See Unit-IV, Page 171, Q.28)

(b) Distinguish between the static and dynamic analysis of a program. Explain at least one metric that a static analysis tool reports and at least one metric that a dynamic analysis reports. How are these metrics useful ? (See Unit-IV, Page 155, Q.5) 10

9. (a) Define the term reverse engineering. Explain the different activities undertaken during reverse engineering. 10 (See Unit-V, Page 221, Q.22)

(b) Discuss the salient features of the organizational reporting structure of the SQA group as recommended by. SEI cmm and ISO 9001. 10

Or

10. Write short notes : 20

(i) Software configuration management (SCM) (See Unit-V, Page 207, Q.5)

(ii) Re-engineering (See Unit-V, Page 214, Q.14)

(iii) Project plan (See Unit-V, Page 227, Q.31)

(iv) Project scheduling. (See Unit-V, Page 245, Q.48)

B.E. (Sixth Semester) EXAMINATION June 2014
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERING AND PROJECT MANAGEMENT
(CS-603)

Note : (i) Attempt one question from each unit. (ii) Each question carry equal marks.

Unit-I

1. (a) What do you understand by term life cycle model of software development ? What problems might a software development organization face if it does not follow any life cycle model during development of a large software product ? (See Unit-I, Page 14, Q.15)

(b) What do you mean by software process ? Write a brief note on component assembly model. (See Unit-I, Page 33, Q.44)

Or

2. (a) What are the major phases in the waterfall model of software development ? Which phase consumes the maximum effort for developing a typical software product ? (See Unit-I, Page 15, Q.17)

(b) What is prototyping model ? Under what circumstances, it is beneficial to construct a prototyping model ? Does construction of a prototyping model always increase the overall cost of software development ? (See Unit-I, Page 19, Q.22)

Unit-II

3. (a) What do you understand by traceability in the context of software requirement specification ? How is traceability achieved ? Why is traceability important considered an important issue ? (See Unit-II, Page 97, Q.50)

(b) What are the different types of requirements gathering activities that the analysis use to gather requirements from a customer ? (See Unit-II, Page 60, Q.9)

Or

4. (a) How are the abstraction and decomposition principles used in developing a good Software Requirements Specification (SRS).

(b) Discuss a brief overview on object oriented software development.

Unit-III

5. (a) What do you mean by terms cohesion and coupling in the context of software design ? How are these concepts useful in arriving at a good design of a system ? (See Unit-III, Page 131, Q.47)

(b) Do you agree with the following assertion ? "A design solution that is difficult to understand would lead to increased development and maintenance cost". Give reasonings for your answer. (See Unit-III, Page 147, Q.68)

Or

6. Write a short notes on (any four) –

(i) User interface design (See Unit-III, Page 119, Q.29)

(ii) Design metrics (See Unit-III, Page 144, Q.65)

(iii) Software modeling (See Unit-III, Page 110, Q.17)

(iv) UML (See Unit-III, Page 111, Q.18)

(v) Function-oriented design. (See Unit-III, Page 125, Q.37)

Unit-IV

7. (a) Supposed a developed software has successfully passed all the three level of testing i.e., unit testing, integration testing and system testing can we claim that the software is defect free ? Justify your answer. (See Unit-IV, Page 187, Q.46)

(b) Write the difference between black-box testing and white-box testing. (See Unit-IV, Page 171, Q.28)

Or

8. (a) What do you understand by system testing ? What are the different kinds of system testing that are usually performed on large software products ? (See Unit-IV, Page 186, Q.45)

(b) Do you agree with the following statement. "System testing can be considered a pure black-box test"? Justify your answer. (See Unit-IV, Page 178, Q.37)

Unit-V

9. (a) What are the different types of maintenance that a software product might need ? Why are these maintenance required ? (See Unit-V, Page 206, Q.4)

(b) What do you mean by the term software reverse engineering ? Why is it required ? Explain the different activities undertaken during reverse engineering. (See Unit-V, Page 221, Q.22)

Or

10. (a) What do you understand by software project management ? How quality of software can be assured ? (See Unit-V, Page 259, Q.67)

(b) Write a brief note on risk assessment. (See Unit-V, Page 253, Q.57)

B.E. (Sixth Semester) EXAMINATION, Dec. 2014
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERING AND PROJECT MANAGEMENT
[(CS-603)]

Note : (i) Attempt one question from each unit. Each unit have equal marks.

(ii) Assume data/value if required.

Unit-I

1. (a) Difference between water fall model and prototype model. (See Unit-I, Page 20, Q.23)

(b) Explain unified process. What are its characteristics ? (See Unit-I, Page 10, Q.10)

Or

2. (a) As you move outward along the process flow path of spiral model what you can say about software that is being developed or maintained ? (See Unit-I, Page 31, Q.39)

(b) Describe about CMM. (See Unit-I, Page 35, Q.49)

Unit-II

3. (a) Difference between structured analysis and object oriented analysis. (See Unit-IV, Page 201, Q.63)

(b) What are the characteristics of good SRS ? (See Unit-II, Page 87, Q.36)

Or

4. (a) Compare functional and behavioural models. (See Unit-II, Page 60, Q.8)

(b) Explain about use case modelling. (See Unit-II, Page 79, Q.29)

Unit-III

5. (a) Discuss function oriented design technique. (See Unit-III, Page 137, Q.59)

(b) Explain user interface design process. (See Unit-III, Page 122, Q.32)

Or

6. (a) What is design ? Discuss the objective of software design. (See Unit-III, Page 99, Q.1)

(b) Explain about component based design.(See Unit-III, Page 140, Q.61)

Unit-IV

7. (a) Difference between the top down and bottom up integration testing. (See Unit-IV, Page 184, Q.42)

(b) What is system testing ? What series of tests are performed during system testing ? (See Unit-IV, Page 185, Q.44)

Or

8. (a) Differentiate static and dynamic testing done state few salient characteristics of modern testing tools. (See Unit-IV, Page 154, Q.4)

(b) Discuss software testing strategies. Differentiate between verification and validation. (See Unit-IV, Page 166, Q.22)

Unit-V

9. (a) What are the activities performed during SQA ? (See Unit-V, Page 259, Q.66)

(b) What do you mean by risk mitigation ? (See Unit-V, Page 253, Q.58)

Or

10. (a) Write short note on – project scheduling.(See Unit-V, Page 245, Q.48)

(b) Describe COCOMO model. (See Unit-V, Page 241, Q.46)

RGPV

B.E. (Sixth Semester) EXAMINATION, June 2015
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERINGAND PROJECT MANAGEMENT
(CS-603)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each questions are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What is Software process ? How the process is different from methodology show by an example ? (See Unit-I, Page 7, Q.7)

(b) List out the Pros and Cons of waterfall model of software development. (See Unit-I, Page 17, Q.18)

(c) What do you mean by Software Complexity ? How the complexity of software is measured ? (See Unit-I, Page 39, Q.54)

(d) Compute the function point value for a project with the following information domain characteristic –

(i)	No. of external inputs	– 32
(ii)	No. of external outputs	– 60
(iii)	No. of external enquiries	– 24
(iv)	No. of internal logic files	–.08
(v)	No. of external interfaces files	– 02

Assume that all complexity adjustment values are average. (See Unit-I, Page 52, Prob.1)

Or

Show why and how software metrics can improve the software process. Enumerate the effects of metric on software productivity. (See Unit-I, Page 46, Q.61)

Unit-II

2. (a) List out the importance of software requirement document in brief. (See Unit-II, Page 86, Q.35)

(b) Define DFD (Data flow diagram). Also, describe the limitations of DFD for performing structural analysis. (See Unit-II, Page 67, Q.17)

(c) Explain the different characteristics of software requirement specification. (See Unit-II, Page 87, Q.36)

(d) What are the elements of analysis modeling explain each of them in brief ? (See Unit-II, Page 73, Q.22)

Or

Why validation is important in the requirement phase ? Justify your answer with proper example. (See Unit-II, Page 95, Q.48)

Unit-III

3. (a) What are the different system views that can be modelled using UML? (See Unit-III, Page 111, Q.19)

(b) Differentiate between cohesion and coupling with example. **(See Unit-III, Page 131, Q.47)**

(c) Define software design process. Describe various steps involve in software design. **(See Unit-III, Page 103, Q.7)**

(d) What are the fundamental principles of user interface design ? Explain and enumerate the end user requirement in user interface design. **(See Unit-III, Page 120, Q.31)**

Or

Why is modularity ? List out the important properties of modular system in brief. **(See Unit-III, Page 132, Q.51)**

Unit-IV

4. (a) What are the primary objectives of glass box testing ?

(b) Calculate the cyclomatic complexity for following program. Also explain your approach –

```
int temp
if (a > b)
    temp – a
else
    temp = b
if (c > temp)
    temp = c
return temp
```

(See Unit-IV, Page 198, Prob.4)

(c) Explain the term boundary value analysis with suitable example. **(See Unit-IV, Page 172, Q.30)**

(d) Define functional testing. Also, explain the various approaches used in functional testing. **(See Unit-IV, Page 175, Q.33)**

Or

Describe integration testing. Explain the steps for top-down integration. Also list out shortcomings of top-down integration. **(See Unit-IV, Page 184, Q.43)**

Unit-V

5. (a) What are the advantages of reverse engineering process ? **(See Unit-V, Page 222, Q.23)**

(b) Justify the importance of feasibility study in software development.

(c) What is Gantt chart ? Give the advantages of Gantt charts in monitoring software project. **(See Unit-V, Page 252, Q.53)**

(d) Describe the components of software maintenance process. Why the cost of software maintenance is high ? **(See Unit-V, Page 204, Q.2)**

Or

Discuss various cost estimation models and compare them. **(See Unit-V, Page 243, Q.47)**

B.E. (Sixth Semester) EXAMINATION, Dec. 2015
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERING AND PROJECT MANAGEMENTS
(CS-603)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What is a model ? What are the benefits with model ? **(See Unit-I, Page 13, Q.13)**

(b) What are the advantages of developing the prototype of a system? **(See Unit-I, Page 22, Q.25)**

(c) What are the characteristics to be considered for the selection of a life cycle model ? **(See Unit-I, Page 18, Q.21)**

(d) How prototype model solve the problems over the waterfall model ? **(See Unit-I, Page 19, Q.22)**

Or

"Traditional software process models do not deal sufficiently with the uncertainty". Which model and how solve this problem ? **(See Unit-I, Page 26, Q.34)**

Unit-II

2. (a) What are the steps involved in requirement engineering ? **(See Unit-II, Page 94, Q.46)**

(b) How many types of requirements are possible and why ? **(See Unit-II, Page 57, Q.4)**

(c) List out requirements elicitation techniques. Which one is most popular and why ? **(See Unit-II, Page 64, Q.13)**

(d) Why is software requirements specification (SRS) need in a software project. And what are advantages with it. **(See Unit-II, Page 89, Q.39)**

Or

What should be parameter and methods to check an SRS document for the requirement validation ?

Unit-III

3. (a) What is a design ? Describe the difference between conceptual design and technical design. **(See Unit-III, Page 100, Q.2)**

(b) What is modularity ? List the important properties of a modular system. **(See Unit-III, Page 132, Q.51)**

(c) Define the module coupling and explain different type of coupling. **(See Unit-III, Page 132, Q.50)**

(d) Explain why it may be necessary to design the system architecture before the specification is written. (See Unit-III, Page 118, Q.27)

Or

If some existing modules are to be re-used in building a new system, which design strategy is used and why ? (See Unit-III, Page 136, Q.57)

Unit-IV

4. (a) What are the error, fault and failure regarding to system ? And why it occurs in system ? (See Unit-IV, Page 158, Q.9)

(b) Explain the boundary value analysis testing technique with the help of an example. (See Unit-IV, Page 172, Q.30)

(c) Gives the three examples where when we apply black box testing we don't find errors but when we apply white box testing we find errors and also in case of vice-versa.

(d) Compute the cyclomatic complexity by all three methods for finding greater number between two variables. (See Unit-IV, Page 197, Prob.3)

Or

What is integration testing technique ? How many types of it ? Explain. (See Unit-IV, Page 181, Q.41)

Unit-V

5. (a) What is version control in project ? (See Unit-V, Page 210, Q.10)

(b) How many types of feasibility analysis apply in a project ? (See Unit-V, Page 226, Q.30)

(c) How COCOMO model work for the cost estimation ? (See Unit-V, Page 241, Q.46)

(d) A project size of 2000 KLOC is to be developed. Software development team has average experience on similar types of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project. (See Unit-V, Page 262, Prob.2)

Consider a project with the following functional unit –
Number of user inputs = 50
Number of user outputs = 40
Number of user enquires = 35
Number of user files = 06
Number of external interfaces = 04
Assume all complexity adjustment factors and weighting factors are averages.
Compute the function points for the project. (See Unit-I, Page 54, Prob.2)

Or

Suppose you are the project manager of a large software development project. List three common types of risks that your project might suffer. Point out the main steps that you would follow to effectively manage risks in your project. (See Unit-V, Page 256, Q.63)

RGPV

B.E. (Sixth Semester) EXAMINATION, June 2016
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERING AND PROJECT MANAGEMENT
(CS-603)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each questions are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) Define software process. (See Unit-I, Page 7, Q.6)

(b) Explain the characteristics of the software. (See Unit-I, Page 3, Q.3)

(c) What is capability maturity model ? (See Unit-I, Page 35, Q.49)

(d) Explain the term software development life cycle model. What are the consequences if an organization does not follow any life cycle model during development of the software product ? (See Unit-I, Page 14, Q.15)

Or

Explain spiral model in detail and under what circumstances is it beneficial. (See Unit-I, Page 26, Q.34)

Unit-II

2. (a) What are functional and non-functional requirement ? (See Unit-II, Page 56, Q.3)

(b) Why requirement elicitation is difficult ? (See Unit-II, Page 62, Q.11)

(c) Explain use case model. (See Unit-II, Page 79, Q.29)

(d) What is the difference between function oriented and object oriented modelling ? Explain in detail. (See Unit-II, Page 78, Q.26)

Or

Explain software requirement specification in detail and why it is necessary. (See Unit-II, Page 89, Q.40)

Unit-III

3. (a) What is coupling ? (See Unit-III, Page 130, Q.46)

(b) What is cohesive module ? (See Unit-III, Page 125, Q.38)

(c) Explain Data Flow Diagram (DFD). (See Unit-II, Page 65, Q.15)

(d) What is design ? Discuss the objective of software design. (See Unit-III, Page 99, Q.1)

Or

Explain user interface design and UML. (See Unit-III, Page 124, Q.35)

Unit-IV

4. (a) What is code inspections ? (See Unit-IV, Page 156, Q.7)

(b) What is equivalence partitioning ? (See Unit-IV, Page 172, Q.29)

(c) What are the uses of testing tools ? (See Unit-IV, Page 194, Q.55)

(d) Write short notes on –

(i) Black-box testing (See Unit-IV, Page 169, Q.25)

(ii) Write box testing (See Unit-IV, Page 169, Q.25)
(iii) Unit testing (See Unit-IV, Page 179, Q.38)

Or

Describe integration testing and what are the approaches for integration testing. (See Unit-IV, Page 181, Q.41)

Unit-V

5. (a) Explain reverse engineering. (See Unit-V, Page 221, Q.20)
(b) What is feasibility analysis ? (See Unit-V, Page 225, Q.28)
(c) What are the objectives and features supported by software configurations management ? (See Unit-V, Page 209, Q.7)
(d) Explain risk assessment and mitigation. (See Unit-V, Page 254, Q.59)

Or

Explain cost estimation COCOMO model with exmples. (See Unit-V, Page 241, Q.46)

RGPV B.E. (Sixth Semester) EXAMINATION, Dec. 2016
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERINGA ND PROJECT MANAGEMENT
(CS-603)

Note : (i) Answer any five questions.
(ii) All questions carry equal marks.
(iii) Assume suitable data if missing.

1. (a) Explain iterative and spiral model of software development. 7 (See Unit-I, Page 31, Q.40)
(b) Describe the evolutionary process model and the model that make use of existing component with example. 7 (See Unit-I, Page 33, Q.43)

2. (a) Explain requirement engineering with complete process. Also give its types. (See Unit-II, Page 57, Q.5) 7
(b) Compare function oriented and object oriented software development. (See Unit-II, Page 78, Q.26) 7

3. Write short notes on – 14
(a) Modularization (b) Coupling cohesion (c) Design verification. (See Unit-III, Page 134, Q.55)

4. (a) Explain the types of software design strategies available. 7 (See Unit-III, Page 135, Q.56)
(b) Briefly describe software analysis and design tools. 7 (See Unit-III, Page 106, Q.12)

5. (a) Compare functional and structural testing approaches. 7 (See Unit-IV, Page 171, Q.28)
(b) What are the aspects considered before software testing while software testing and after software testing. 7

6. (a) What are the various software maintenance activities involved throughout its cycle ? (See Unit-V, Page 206, Q.4) 7

(b) How reverse engineering is different from re-engineering process ? Explain with suitable example ? (See Unit-V, Page 221, Q.21) 7

7. (a) Compare object oriented analysis of software with that of structural software engineering approach. (See Unit-IV, Page 201, Q.63) 7
(b) Describe various program comprehension techniques. 7

8. Describe the followings – 14
(a) Test Metrics (See Unit-IV, Page 192, Q.53)
(b) Agile Process (See Unit-I, Page 35, Q.47)
(c) Software Quality Assurance (SQA). (See Unit-V, Page 257, Q.65)

RGPV B.E. (Sixth Semester) EXAMINATION, June 2017
(Computer Science & Engg. Branch)
SOFTWARE ENGINEERINGA ND PROJECT MANAGEMENT
(CS-603)

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. What is the importance of models in software engineering ? Explain with examples any two process models which are commonly used. (See Unit-I, Page 24, Q.30)

2. Explain the various stages of WINWIN Spiral model. (See Unit-I, Page 29, Q.36)

3. Discuss an example of type of system where social and political factors might strongly influence the system requirements. Explain why these factors are important in your example ? (See Unit-II, Page 83, Q.32)

4. Explain the steps involved in Object Oriented modeling. (See Unit-II, Page 77, Q.25)

5. Describe the metrics for the design model of a product. What are the attributes of effective software metrics ? (See Unit-III, Page 144, Q.66)

6. Explain the process of User Interface design. Also provide guideline for Good user Interface. (See Unit-III, Page 124, Q.34)

7. What is overall strategy for software testing. Explain it clearly. (See Unit-IV, Page 165, Q.21)

8. Answer any four of the following –
(a) Explain the process maturity levels in SEI's CMM. (See Unit-I, Page 35, Q.49)
(b) Explain the nature of SRS ? (See Unit-II, Page 87, Q.36)
(c) How do you assess an architectural style that has been derived ? (See Unit-III, Page 115, Q.25)
(d) The software analysis and design are constructive task and software testing is considered to be destructive from the developer point of view. Discuss. (See Unit-IV, Page 159, Q.13)
(e) What are the problems encountered with uncontrolled change management ? (See Unit-V, Page 210, Q.9)

(f) Software project scheduling does not differ from scheduling of any other multitask engineering project. Discuss. (See Unit-V, Page 250, Q.51)

RGPV

CS-603 (GS)
B.E. (VI Semester) EXAMINATION, Dec. 2017
Grading System (GS)
SOFTWARE ENGINEERING & PROJECT MANAGEMENT

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Explain in detail about the life cycle process. 7 (See Unit-I, Page 13, Q.14)
   (b) Explain the prototyping approaches in software process. 7 (See Unit-I, Page 20, Q.24)
2. (a) What is meant by cohesion ? How should a software be designed considering cohesion ? (See Unit-III, Page 130, Q.45) 7
   (b) Lit out the activities involved in software requirement analysis. What is requirement validation ? (See Unit-II, Page 60, Q.7) 7
3. (a) Discuss the core activities involved in user interface design process. 7 (See Unit-III, Page 124, Q.33)
   (b) Write a note on architectural design. (See Unit-III, Page 112, Q.20) 7
4. (a) Write in detail about the object oriented system design and also discuss about UML. (See Unit-IV, Page 200, Q.59) 7
   (b) What is integration testing ? Discuss about the various approaches of integration testing. (See Unit-IV, Page 181, Q.41) 7
5. (a) What are the fundamentals of software testing and discuss the characteristics of a good test ? (See Unit-IV, Page 161, Q.15) 7
   (b) Brief the software configuration management and its process. 7 (See Unit-V, Page 207, Q.5)
6. (a) Explain the primary responsibility of software quality assurance group. (See Unit-V, Page 257, Q.64) 7
   (b) What is software quality assurance ? What are the measures of software quality ? (See Unit-V, Page 259, Q.68) 7
7. (a) Discuss the tools and techniques used in project management. 7 (See Unit-V, Page 229, Q.32)
   (b) Explain black-box testing and white-box testing. 7 (See Unit-IV, Page 169, Q.25)
8. (a) Explain the concept of modularity in design engineering process. 7 (See Unit-III, Page 133, Q.52)
   (b) Write any three requirements definition document. 7 (See Unit-II, Page 90, Q.43)

RGPV

CS-6003 (CBGS)
B.E. (VI Semester) EXAMINATION, May 2018
Choice Based Grading System (GS)
SOFTWARE ENGINEERING & PROJECT MANAGEMENT

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Define software engineering. Explain software engineering – A layered technology. (See Unit-I, Page 5, Q.4) 7
   (b) Explain linear sequential model for software development. 7 (See Unit-I, Page 15, Q.17)
2. (a) Define software process. Explain RAD model for software development with its various phases. (See Unit-I, Page 24, Q.29) 7
   (b) Explain software requirement specification. 7 (See Unit-II, Page 86, Q.34)
3. (a) Explain acitivities covered by the software project management. 7 (See Unit-V, Page 224, Q.27)
   (b) Explain the various SDLC activities as outlined by ISO 12207 with a neat diagram. (See Unit-V, Page 231, Q.35) 7
4. (a) What are the dimensions of requirement gathering ? 7 (See Unit-II, Page 60, Q.9)
   (b) Differentiate between function-oriented and object-oriented software development. (See Unit-II, Page 78, Q.26) 7
5. (a) What is software design ? Explain various principles and design concepts of software design. (See Unit-III, Page 109, Q.15) 7
   (b) What is functional independence ? Explain different cohesion and coupling types in detail. (See Unit-III, Page 132, Q.49) 7
6. (a) What is software requirement specification (SRS) ? State its principles and characteristics. (See Unit-II, Page 86, Q.34) 7
   (b) How software inspection improves software quality ? Explain software inspection process in brief. (See Unit-IV, Page 166, Q.23) 7
7. (a) What are the various quality concepts of SQA ? (See Unit-V, Page 257, Q.65)
   (b) What is SCM ? Explain the concept of baseline and SCM items in brief. (See Unit-V, Page 233, Q.37) 7
8. Explain the following –
   (a) Test case design (See Unit-IV, Page 163, Q.19)
   (b) Integration testing (See Unit-IV, Page 181, Q.41)
   (c) Black-box testing (See Unit-IV, Page 170, Q.26)
   (d) Test matrics. (See Unit-IV, Page 192, Q.53)

RGPV

CS-403 (CBGS)
B.Tech., IV Semester
EXAMINATION, May 2019
Choice Based Grading System (CBGS)
SOFTWARE ENGINEERING

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Explain software life cycle of spiral model and discuss various activities in each phase. **(See Unit-I, Page 28, Q.35)** 7

   (b) Explain about software engineering paradigm in detail. 7 **(See Unit-I, Page 15, Q.16)**

2. (a) Explain in detail about the software process. 7 **(See Unit-I, Page 9, Q.9)**

   (b) Explain the ways and means for collecting the software requirements.7 **(See Unit-II, Page 63, Q.12)**

3. (a) Describe various software prototyping techniques. 7 **(See Unit-I, Page 20, Q.24)**

   (b) Differentiate between function-oriented and object-oriented software development. **(See Unit-II, Page 78, Q.26)** 7

4. (a) Explain the use case diagram of ATM machine with neat diagram. 7 **(See Unit-II, Page 81, Q.30)**

   (b) Explain about the various design concepts considered during design. 7 **(See Unit-III, Page 107, Q.13)**

5. (a) Explain architectural and procedural design for a software. 7 **(See Unit-III, Page 112, Q.21)**

   (b) Describe the golden rules for interface design. 7 **(See Unit-III, Page 119, Q.30)**

6. (a) Discuss the differences between black box and white box testing models. **(See Unit-IV, Page 171, Q.28)** 7

   (b) What do you mean by system testing ? Explain in detail. 7 **(See Unit-IV, Page 185, Q.44)**

7. (a) What do you mean by boundary value anlaysis ? Give two examples of boundary value testing. **(See Unit-IV, Page 172, Q.30)** 7

   (b) Discuss briefly on software maintenance activities. 7 **(See Unit-V, Page 206, Q.4)**

8. (a) Describe two metrics which are used to measure the software in detail. **(See Unit-I, Page 47, Q.62)** 7

   (b) List and explain the steps in risk management process. 7 **(See Unit-V, Page 256, Q.62)**

●●●